DSTO

Information Technology Division

RESEARCH REPORT
ERL-0800-RR

MODULAR IMPLEMENTATION OF FEATURE EXTRACTION AND MATCHING ALGORITHMS FOR PHOTOGRAMMETRIC STEREO IMAGERY

by

James Kershaw and Garry Hamlyn

ELECTRONICS RESEARCH LABORATORY

APPROVED FOR PUBLIC RELEASE

AR-008-809

DSTO AUSTRALIA

ELECTRONICS RESEARCH LABORATORY

Information Technology Division

RESEARCH REPORT
ERL-0800-RR

MODULAR IMPLEMENTATION OF FEATURE EXTRACTION AND MATCHING ALGORITHMS FOR PHOTOGRAMMETRIC STEREO IMAGERY

by

James Kershaw and Garry Hamlyn

SUMMARY

This paper describes the implementation of algorithms for automatically extracting and matching features in stereo pairs of images. The implementation has been designed to be as modular as possible to allow different algorithms for each stage in the matching process to be combined in the most appropriate manner for each particular problem. The modules have been implemented in the AVS environment but are designed to be portable to any platform.

This work has been undertaken as part of task DEF 93/163 "Intelligence Analysis of Imagery", and forms part of ITD's contribution to the Visual Processing research program in the Cooperative Research Centre for Sensor Signal and Information Processing (CSSIP). A major aim of both the task and the research program is to produce software to assist intelligence analysts in extracting three dimensional shape from imagery: the algorithms and software described here will form the first part of a module for automatically extracting depth information from stereo image pairs.

JUNE 94

APPROVED FOR PUBLIC RELEASE

POSTAL ADDRESS: Director, Electronics Research Laboratory, PO Box 1500, Salisbury, South Australia, 5108.

CONTENTS

FIGURES

APPENDICES

THIS IS A BLANK PAGE

1 INTRODUCTION

This report describes software developed as part of a major research program to produce systems to assist analysts in reconstructing size and shape from imagery such as satellite images or photographs. Shape reconstruction can aid the intelligence community in a variety of tasks, for example compiling technical intelligence on new systems or developing virtual reality scenarios such as fly throughs. In particular the ability to determine the shape of new objects ranging from buildings through to ship propellers is of great help in determining the properties or functions of such objects. In this area the development of computer packages to automate various tasks in photogrammetry would significantly increase the productivity of photogrammetric intelligence analysis. Therefore, as part of DSTO's support to the Defence Intelligence Organisation (DIO) and other image analysis centres in the ADF, ITD has undertaken a research program to develop state-of-the-art photogrammetric software tailored to the needs of intelligence analysts. At present this work is being carried out under tasks DEF 93/163 "Intelligence Analysis of Imagery" and NAV 93/071 "Image Processing and Analysis for Navy". The research program is located within the Visual Processing program in the Cooperative Research Centre for Sensor Signal and Information Processing (CSSIP) and forms part of ITD's contribution to CSSIP.

The particular work reported here tackles a major subtask in photogrammetry: the reconstruction of shape from a pair of images. At present the task of extracting three dimensional shape information from two spatially disparate images of the same target scene is not fully automated. Existing systems for solving this problem are only semi-automatic and require the user to specify matched points and camera parameters. The aim of the current research program is to minimise the amount of user interaction or a-priori information needed to match the two images. To date there has been no reliable implementation of a single robust algorithm for automatic matching that performs well on a broad range of image types of interest to analysts. Moreover, while a number of algorithms for automatic matching in particular scenarios have been described in the research literature, little has been done in the way of systematic tests to verify their actual performance.

Given this situation the Visual Processing program has decided to assemble within a common environment a number of promising algorithms from the literature for each of the various stages in stereo photogrammetry. These algorithms will then be tested separately and in combination to investigate the individual performance of each algorithm and to identify the best way to fuse the algorithms together in a final system. This report describes progress to date on part of this task: in particular it describes and documents the implementation of four algorithms (Moravec, Fex, Barnthom and Hannah) that can be used in a modular and interchangeable way to carry out the first two steps of the automatic stereo photogrammetry, namely feature extraction and feature matching.

The structure of the report is as follows. Section two introduces the data representations and the way that the early layers of the algorithm interact. Sections three, four and five describe the feature extraction algorithms, while sections six, seven and eight outline the feature matching algorithms. The present status of the work is summarised in section nine. Appendix A summarises the form of the common data structures used by the various routines, while user documentation on each routine is found in Appendix B.

The modules described in this report are implemented under AVS (Application Visualization System), an image processing and visualisation package produced by Advanced Visual Systems [1]. Nevertheless, while the algorithms described here have been developed in AVS and with stereo photogrammetry in mind, their implementation has been made as modular and self-contained as possible so that the routines can be ported to different environments or used on other image analysis tasks. Indeed they may also be useful for more general feature matching tasks in data fusion or similar applications. In particular the Hannah algorithm described here has already been ported to the KHOROS image processing system [12] and incorporated in software for automatic image registration that is concurrently being developed in the Visual Processing program (see [2]).

2 MODULAR STRUCTURE IN FEATURE MATCHING

While the ultimate goal of the Visual Processing program is to produce a general software package for shape reconstruction, such a package will necessarily consist of a collection of specialist routines for particular tasks, such as reconstruction of shape from perspective distortion in a single image or from relative displacements in a stereo pair of images. Each task can in turn be broken up into various stages consisting of subtasks, with output from one stage being passed as input to the next stage. Many of these subtasks are common to a number of different larger tasks, so in the long run it is most efficient to try and identify the modular structure implicit in existing or new algorithms for shape reconstruction as this will allow code reuse and quicker construction of algorithms. If such structure can be identified, the algorithms can then be implemented as a series of self-contained routines linked through standardised data structures.

In particular shape reconstruction from stereo imagery is almost invariably performed in two stages: in the first stage significant features (e.g. corners or edges) of the three dimensional (3D) object being imaged are located in both images; while in the second stage the movements of the features across the two images is used to infer their actual location in space by triangulation. The first stage in turn can be split into two subtasks. The first is feature detection, in which interesting features in each image are identified. The expectation is that each such feature in an image reflects a corresponding significant feature on the underlying object, and so will also be present in the other image. The second subtask involves matching the interesting features in one image with those in the other image.

It is, of course, possible to write algorithms that carry out both feature detection and feature matching in a manner that cannot be easily separated. For example, feedback from the

matching stage can be used to refine feature identification as in the Hannah matching algorithm described in section 8. Nevertheless the benefits of taking advantage of any modular structure are sufficiently great to warrant separating algorithms into self-contained modules where possible.

One particular benefit of modular development is that it both clarifies the requirements on data structure and allows greater freedom in their construction. After some discussion, the Visual Processing program has settled on the following approach. First, internal data structures in individual modules are specified independently so as to give the most efficient structure for internal use. Second, standard AVS data structures have been specified for external transfer of data between modules and to the AVS interface; the formats of these structures have been chosen for ease of use within AVS. Finally, interfaces (or "wrappers") are written around each module that translate between the internal and external data structures.

When applied to feature matching, the above philosophy has led to the following task and data structure breakdown. First, two modular subtasks have been identified: feature extraction and feature matching. This in turn means that there are three points at which common data structures have to be enforced: at input; after feature extraction and before feature matching; and after feature matching. Figure 1 shows the conceptual arrangement of the modules and the data structures.

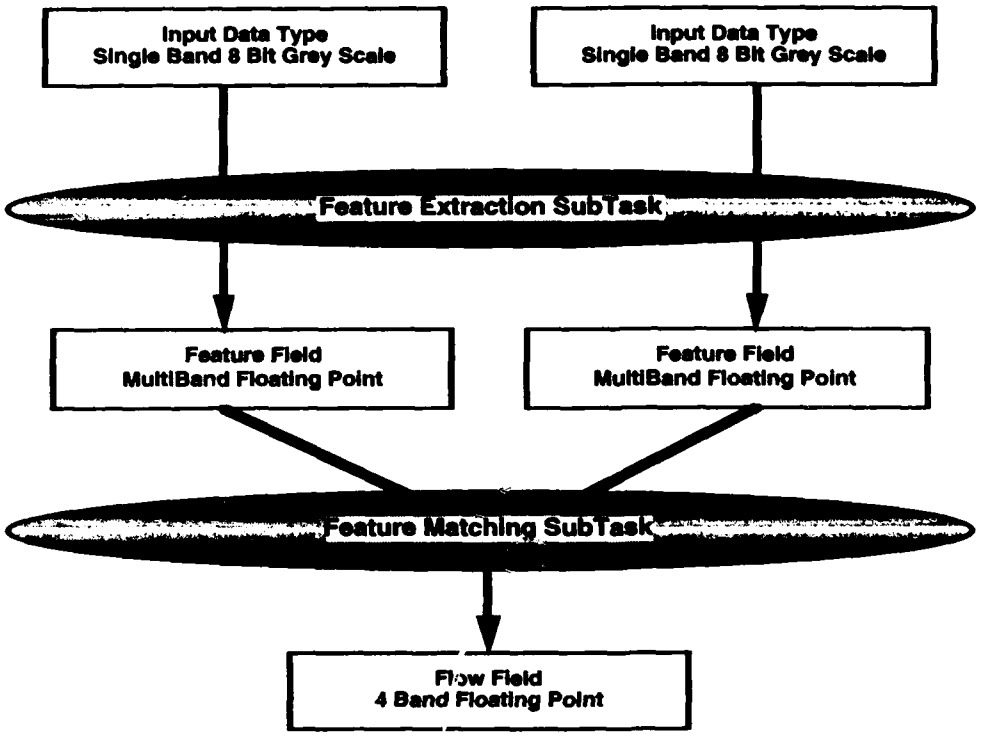


Figure 1 : Module and Layer Interaction

2.1 General Comments on Implementation

This subsection outlines implementation aspects that are common to all modules, and should be read in conjunction with the implementation notes provided (in later sections) for individual modules.

2.1.1 Source Code

Individual modules have been implemented as far as possible as standard Kernighan and Ritchie (K & R) C programs [9], using the C compiler bundled with SunOS 4.1.3.

All AVS-specific code has been excluded from each "core" module to improve portability. To this end, AVS-specific interfaces are provided to handle the input and output and make the necessary conversions to and from generic C data structures (for example, images are converted from AVS fields to standard 2D or 3D arrays as appropriate). Although this practice introduces overheads in run time and computer memory usage, the authors believe that the improvement in code portability outweighs any disadvantages.

It is proposed to use the Unix Source Code Control system (SCCS) as a configuration management tool for all AVS shape inference code, however arrangements for this have not yet been formalised.

2.1.2 Operating System and Platform

Each module has been successfully compiled and run on a Sun SPARCStation 10 under SunOS 4.1.3.

2.1.3 File System Structure

All source and executable files for the modules are contained in subdirectories under the *ii3:/gig3c/users/avs* directory on the ITD network. This disk (and server) is physically located in the Image Information (II) Section of Information Management (IM) Group, ITD in 79 Labs Salisbury. For a listing of all the relevant files for each module please refer to the detailed module descriptions later in this report.

2.1.4 AVS Data Structures

The feature extraction modules output one feature field for each input image. The feature field is represented within AVS as an AVS 2D field of vectors of floating point values, alternatively it can be thought of as a multi band image in which each band is a collection of floating point values. The field contains a reproduction of the original image in the first band. If no feature has been detected at a given pixel, the remaining bands contain zeros,

otherwise they contain a list of the attributes of the feature at this location. The feature matching module takes in two feature fields and returns a single flow field which represents the relation between points in the two feature fields. The flow field is an AVS 2D field of vectors of four floating point values (or again an image of four floating point bands). The first two bands contain representations of the two original images. If a pixel location in the first image has no matched feature then the third and fourth bands contain a value representing "No Match", otherwise they contain the vertical and horizontal displacements from the particular pixel to the location of the corresponding feature in the second image. A summary of these formats is given in Appendix A.

2.1.5 Implementing User Defined Functions as AVS Modules

AVS requires at least three functions to enable a user defined function to be executed under the AVS system. These are an **initialisation** function which tells AVS which modules are included in the code block, a **description** function which identifies the module to AVS and declares its name, I/O ports and parameters, and a **computation** function which actually performs the computational work of the module, using the input data and parameters to produce output of some sort. AVS invokes this function when the AVS flow executive is active and when the module's input data or parameters change. In the implementation covered in this report the computation function performs the necessary conversions to and from generic C structures and calls the user supplied function(s).

2.1.6 Customised "ShapeInference" AVS Module Library

A customised module library (named "ShapeInference") for use with the AVS Network Editor module palette has also been created. It is intended that all AVS shape inference modules developed should be added to this library as they are completed. The description file for the library is *ii3:/gig3c/users/avs/module_library/ShapeInference*. In order for this to be loaded automatically at startup, AVS users should create a file called *.avsrc* in their home directory and insert the line

```
ModuleLibraries
/gig3c/users/avs/module_library/ShapeInference
```

(note that this will override the loading of the default module libraries - these must also be listed if the user wishes to have them made available as well).

2.1.7 On-line Documentation

On-line documentation has also been provided, and can be accessed in the same manner as standard AVS documentation (bring up the Module Editor window by clicking with the middle or right mouse button on the small square

on the module icon; then click on the Show Module Documentation button). To ensure that the help file is located correctly, the user needs to either set the Unix environment variable AVS_HELP_PATH to */gig3c/users/avs/help*, or alternatively include the line

```
HelpPath    /gig3c/users/avs/help
```

in the *.avsrc* file in their home directory. A copy of each module's on-line help text file is contained in Appendix B

3 FEATURE EXTRACTION

Feature extraction consists of identifying the position and properties of features in an image such as points, lines or regions. The idea is that these image features correspond to features of the underlying 3D scene which will be identifiable in both images. If this is true, then point matching can be performed and the true location in space of the underlying features can be obtained by triangulation. Each feature has an attached vector giving its position and associated properties. For instance, a point might be characterised by its difference from its neighbourhood, a line may have the properties of length, angle, curvature and width, and a region might have values of texture, area, circumference and orientation.

To date two different feature extraction modules have been implemented. They are a point based feature extractor (Moravec, described in section four) which finds feature points, and an edge based feature extractor (FEX, described in section five) which extracts straight line segments. The modules are independent and interchangeable, although, of course, they cannot be mixed together arbitrarily. For example, the left and right feature extraction modules must both return Moravec features, or FEX features, but cannot return one of each type. As described in the previous section, the extraction modules take in a single grey level image and return a floating point feature vector of indeterminate length. Note that the vector length is predefined for each feature extractor, but the generic feature format does not constrain the length of feature vectors.

4 MORAVEC FEATURE EXTRACTION MODULE

The Moravec feature extraction module (henceforth called Moravec) is based on the Moravec feature operator described in [11].

4.1 Description of Moravec Features

Moravec features are point based features determined by the "Moravec interest" of a point. The interest is a measure of how much the point stands out from its local neighbourhood in a grey scale image. The two extreme cases are a white pixel on a black background or a black pixel on a white background, which have the maximum interest value, while a pixel which is exactly the same shade as its local

neighbourhood, or even a line passing through the neighbourhood, has zero interest which is the minimum value. Interest values are calculated by the following local interest operator:

$$M(x,y) = \min \begin{cases} \sum_{i=-2}^{2} (I(x,y) - I(x,y+i))^2 \\ \sum_{i=-2}^{2} (I(x,y) - I(x+i,y))^2 \\ \sum_{i=-2}^{2} (I(x,y) - I(x+i,y+i))^2 \\ \sum_{i=-2}^{2} (I(x,y) - I(x+i,y-i))^2 \end{cases}$$

where $I(x,y)$ is the image point grey level value at the pixel location (x,y) and $M(x,y)$ is the Moravec interest value at the point (x,y). A five by five pixel neighbourhood has been chosen for this module.

Once interest values have been calculated for every point in the image, an initial collection of features are formed by setting the interest value to zero for all points which are not local maxima of the interest function. A user specified number of feature points are then selected from the remaining feature points by retaining those with the highest interest values.

4.2 Implementation in AVS

4.2.1 General

Refer to subsection 2.1 for general information on source code, operating system, file systems, data structures and AVS module structure.

4.2.2 Source Code and File Systems

The source file for this module together with a Makefile are contained in the directory *ii3:/gig3c/users/avs/modules_src/moravec* on the ITD network. The relevant files are:

```
Makefile.avs_moravec
avs_moravec.c
```

The binary executable *avs_moravec* is located in the directory *ii3:/gig3c/users/avs/modules_bin.*

4.2.3 Data Structures

4.2.3.1 AVS

The AVS field data structure is used to pass both image and feature information between modules. These fields are basically two dimensional arrays whose individual elements are vectors of some fixed length. The input data structures are basic images (each element contains one value only - the grey level). The output data structures are feature fields with each element in the two dimensional array containing two values, where the first value is the pixel value in the original image at this location, and the second value is the Moravec interest value at this point. For more information see Appendix A and Appendix B.

4.2.3.2 Other

Within the functional section of the module data is handled using standard C constructs. Given the nature of the Moravec interest operator, the notion of a two dimensional array with a vector at each position has been preserved. The implementation of this is actually as a triply indirected pointer (referencing dynamically allocated memory) which is equivalent to a three dimensional array. The first two dimensions hold the two dimensional data array while the third dimension allows access to individual elements of the vector. Floating point values are used for all calculations. The declaration and method of access is described:

Declaration:	int ***Data;
Access:	Data[i][j][k]
(equivalent to)	*(*(*(Data+i)+j)+k)

4.2.4 Function Descriptions

The following descriptions are not intended to be an exhaustive documentation of functional interactions and dependencies, rather they should be used as a guide to understanding and maintaining the module.

Figure 2 gives an hierarchical description of the functions and their dependencies.

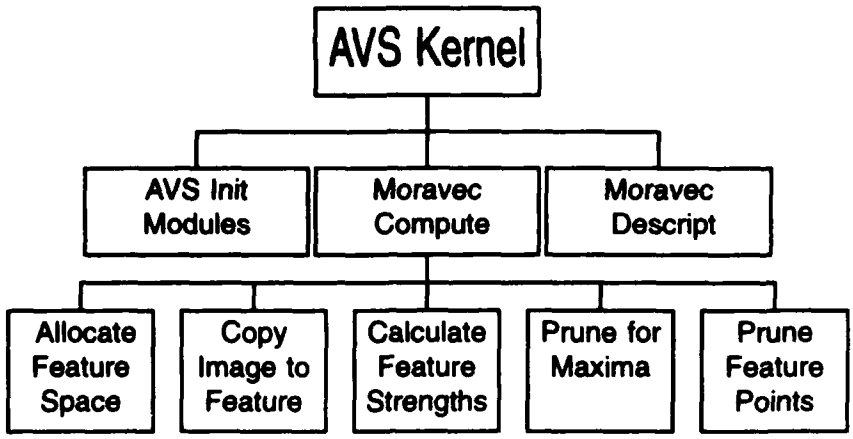


Figure 2: Hierarchical Functional Block Diagram for Moravec

Moravec_Compute

source code file:	avs_moravec.c
called by:	AVS kernel
calls to:	Allocate_Feature_Space
	Copy_Image_to_Feature
	Calculate_Feature_Strengths
	Prune_For_Maxima
	Prune_Feature_Points
	AVS routines

Description:

This is the computation function for the AVS module and is the entry point for the user supplied code.

Allocate_Feature_Space

source code file:	avs_moravec.c
called by:	Moravec_Compute
calls to:	AVS routines
	malloc

Description:

This routine allocates the required space for the output feature field(s) and also allocates space for the internal C-standard representation. It also calculates the maximum number of feature points as defined by the operator.

Copy_Image_to_Feature

source code file: avs_moravec.c
called by: Moravec_Compute
calls to: AVS routines

Description:

This routine copies the input image(s) to the first band of the output field and the first band of the working feature set.

Calculate_Feature_Strengths

source code file: avs_moravec.c
called by: Moravec_Compute
calls to: None

Description:

This routine calculates the Moravec interest value at each point in the working image and stores that value in the second band. The values are then normalised to the interval *[0,1]* by dividing by the maximum value. The actual operator is described in subsection 4.1.

Prune_For_Maxima

source code file: avs_moravec.c
called by: Moravec_Compute
calls to: None

Description:

This routine scans the Moravec interest values and sets all values to zero that are not a local maxima. This is done by comparing the current value with those in the pixel's immediate neighbourhood. If any other point in the neighbourhood is bigger, then the current value is set to zero.

Prune_Feature_Points

source code file: avs_moravec.c
called by: Moravec_compute
calls to: None

Description:

This routine performs an iterative search for a threshold interest value which leaves the specified number of feature points (to within a five percent margin). The method involves toggling interest values between positive for above threshold and negative for below threshold and counting the number of positive feature points. The iterative search for the correct threshold is limited in duration and is basically the bisection method. Finally, all negative values are set to zero, leaving only the positive selection of interest points.

4.3 Example Network and Output

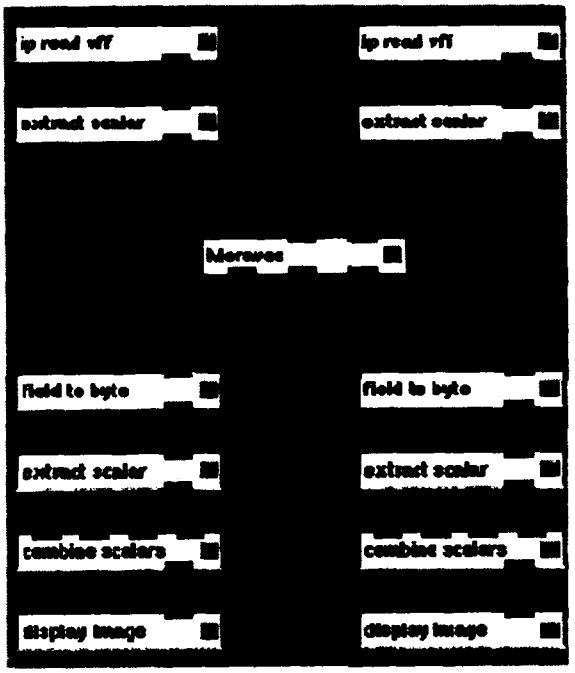


Figure 3 : Example AVS Network for Moravec

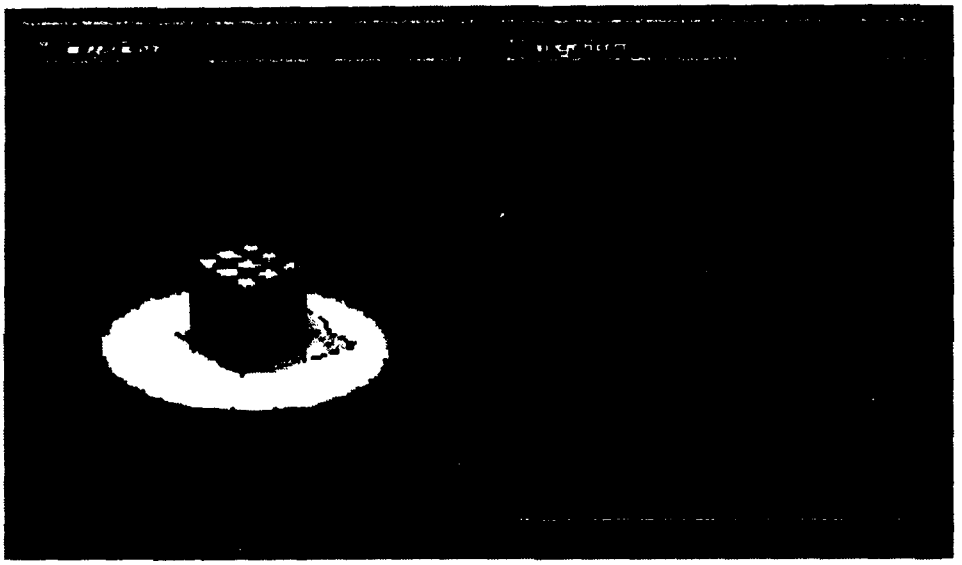


Figure 4 : Example Output for Moravec. The Moravec interest value of selected "interesting" pixels in the input image (left) is indicated by variation in output pixel intensity (right).

4.4 Comments on Parameter Values and Performance

There are two user selectable parameters associated with this module (a detailed explanation of these appears in Appendix B).

In general the Moravec operator extracts reasonable features; typical values for the parameter **FPs (fraction)** should range from 0.0001 to 0.05 in order to extract between 20 and 8000 features for a 512*512 image.

4.5 Future Directions

Further changes to the algorithm may include a change of output format to represent a list of features. A user definable window size for the Moravec neighbourhood may also be considered.

5 FEX FEATURE EXTRACTION MODULE

The second feature extraction algorithm implemented is the edge based feature extraction process (hereafter referred to as FEX) described in [8]. The implementation is based on the code written by A. Etemadi [5] as part of his "perceptual groupings" suite.

5.1 Description of FEX

FEX is an algorithm for extracting line segments from an edge segmented image. The edge segmented image is the result of any line or edge detection algorithm which leaves thinned edges after the filtering process, where thinned edges have a width of only one pixel. Examples of such edge filters are Canny's edge filter [4] and Sobel's edge filter [13]. This process is performed externally to the FEX module.

The edge pixels that remain after the edge filtering process are collected into groups of minimally connected pixels called chains by following edges along touching edge pixels, removing all but one branch pixel at a junction. The extra branches then become their own individual chain. The actual method used is to label each pixel in a string of pixels to a unique string identifier. At a junction the algorithm follows only one of the paths (which path is not important), labelling each pixel. The branch which has not been so labelled will be detected later and given its own unique labelling, stopping when it reaches the branch point where connecting pixels already have a label.

These chains of pixels are processed into straight line segments by starting at one end of the chain and incrementally increasing the number of pixels in the segment until the maximum perpendicular displacement between the segment and the secant passing through the end points of the segment passes some threshold. Once this threshold is

passed the segment is complete and the last point becomes the first point of the next segment.

These segments are then characterised by three parameters (a length and two variance values relating to the straightness of the line segment), which are stored in the last three bands of the output feature array.

5.2 Implementation in AVS.

5.2.1 General

FEX is an algorithm which was down loaded via ftp with minimal changes being made to it in order to enable it to be invoked from AVS. Some of the simpler subroutines were recoded to work on the array based format, but the extraction and classification of segments was left in the original code, with conversion routines written to enable communication between the modules working on arrays and the modules working on lists.

Refer to subsection 2.1 for general information on source code, operating system, file systems, data structures and AVS module structure.

5.2.2 Source Code and File Systems

The source files together with a Makefile are contained in the directory *i3:/gig3c/users/avs/modules_src/fex* on the ITD network. The relevant files are:

```
Makefile.avs_fex
AddElmList.c
CircleLeastSq.c
CopyList.c
CreatList.c
CreateStringList.c
DestList.c
ElmNumList.c
EmptyList.c
FEX.c
LineLength.c
LineSegColIntercept.c
LineSegRowIntercept.c
LineSegTheta.c
LinkSegments.c
PtLinePerpIntercept.c
SizeList.c
StringToSegments.c
```

```
avs_fex.c
FEX.h
LPEG.h
Liste.h
ListeMacros.h
ListeP.h
```

The binary executable avs_fex is located in the directory *ii3:/gig3c/users/avs/modules_bin*.

5.2.3 Data Structures

5.2.3.1 AVS

The AVS field data structure is used to pass both image and feature field information into and out of this module (refer subsection 2.1.4 and Appendices A and B). The input data structures are basic images (each element contains one value only - the grey level). The output data structures are feature fields with each element in the two dimensional array containing four values, where the first value is a reproduction of the original image and the other values are the FEX characteristic parameters for edges.

5.2.3.2 Other

Within the functional section of the module data is handled using standard C constructs as outlined in subsection 4.2.3.2. Within Etemadi's original code the data is stored mainly as list structures which are described in the header files.

5.2.4 Function Descriptions

The following descriptions are not intended to be an exhaustive documentation of functional interactions and dependencies, rather they should be used as a guide to understanding and maintaining the module.

Figure 5 gives an hierarchical description of the functions and their dependencies.

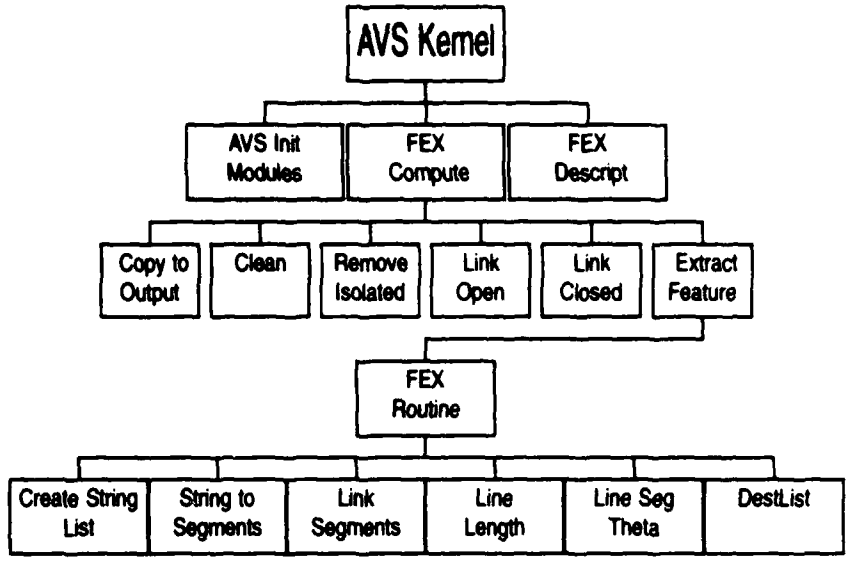


Figure 5 : Hierarchical Functional Block Diagram for FEX

Fex_compute

source code file: avs_fex.c
called by: AVS kernel
calls to: Copy_to_Output
Clean
Remove_Isolated
Link_Open
Link_Closed
Extract_Feature

Description:

"Fex_compute" is the computation function (refer subsection 2.1.5) for this AVS module, and as such is the entry point for the user supplied code.

Copy_to_Output

source code file: avs_fex.c
called by: fex_compute
calls to: AVS routines

Description:

This routine allocates memory for the output space and working space, and copies the image data into the first two bands of each.

Clean
source code file: avs_fex.c
called by: Fex_compute
calls to: None

Description:

This routine places zeros around all of the borders of the images and reduces the thinned edges (as input) to their eight connected minimum.

Remove_Isolated
source code file: avs_fex.c
called by: Fex_compute
calls to: None

Description:

This routine scans the edge image for any isolated edge point and removes them (by writing zero to that position).

Link_Open
source code file: avs_fex.c
called by: Fex_compute
calls to: None

Description:

This routine follows connected chains of pixels which have a distinct end point. Each point in the chain is given a value corresponding to the unique identifier of that chain. Chains containing less than the specified number of pixels are removed.

Link_Closed
source code file: avs_fex.c
called by: Fex_compute
calls to: None

Description:

This is the same as Link_Open, although this handles the case of closed pixel chains as well.

Extract_Feature
source code file: avs_fex.c
called by: Fex_compute
calls to: Fex_Routine

Description:

This is a jumping off point to the functional part of the algorithm

Fex_Routine

source code file:	FEX.c
called by:	Extract_Feature
calls to:	CreateStringList
	StringToSegments
	LinkSegments
	LineLength
	LineSegTheta
	DestList

Description:

This routine performs the actual calculation of location and values of the edge features. It is a slight modification of the original code by Etemadi, with the main difference being the call to CreateStringList which has been modified to convert the array based information to a list based representation native to Etemadi's code as well as performing its original function (to create the string lists). This routine also modifies the output or reporting stage by writing the feature vectors back to the working array.

5.3 Example Network and Output

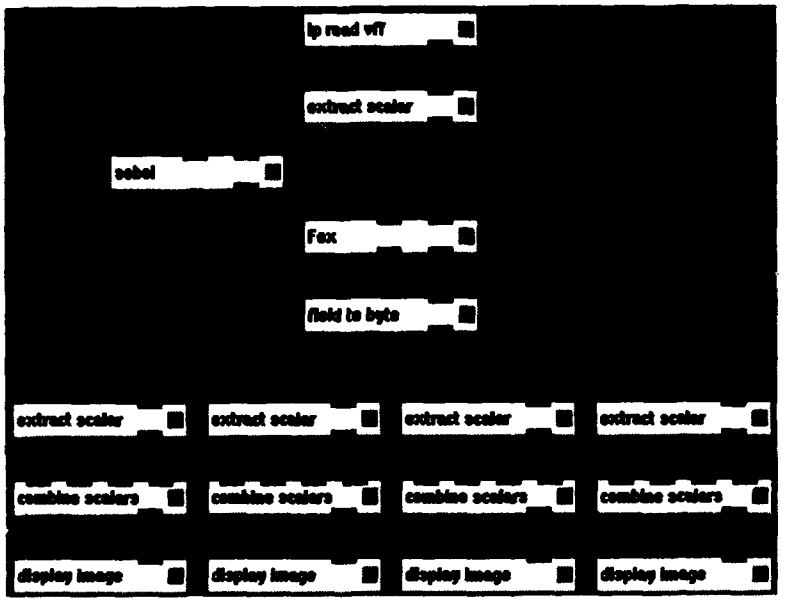


Figure 6 : Example AVS Network for FEX

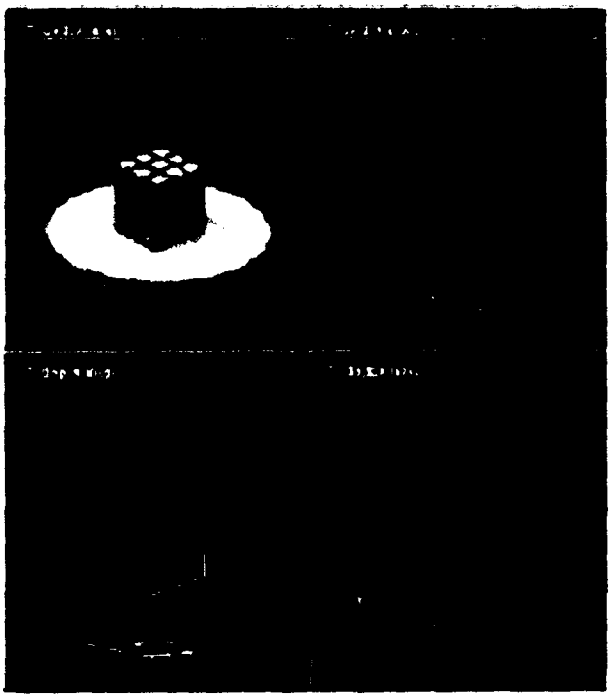


Figure 7 : Example Output for FEX. The first band (top left) contains a copy of the input image while the remaining bands contain the extracted edge segment parameter values.

5.4 Comments on Parameter Values and Performance

There are two user selectable parameters associated with this module (a detailed explanation of these appears in Appendix B).

Reasonable values for parameters **Threshold** and **Min Length** are about 50 and 10 respectively. As expected the number of features extracted depends very much on the characteristics of the image.

5.5 Future Directions

Further changes to the algorithm may include a change of output format to represent a list of features.

6 FEATURE MATCHING

Feature matching involves taking the sets of feature vectors of both images and attempting to match each feature in one image to its corresponding feature in the other image. The result is a set of points that correspond to each other, or a set of points with the appropriate

displacement to its matching point in the other image. There are, however, a number of problems to overcome in achieving this feature matching: features may exist in one image and not in the other due to occlusion of the feature by other objects; distortion due to a change of aspect or perspective can occur; lighting conditions may be different, or the objects might have a high degree of spectral reflection making the perceived intensity dependent on viewing position; lens distortion may not be appropriately compensated at all points in the focal plane, and lens distortion may be dependent on the position of the feature.

At this stage the software suite contains only one matching algorithm capable of handling the generic feature format. This algorithm is the relaxation scheme proposed by Barnard and Thompson [3] and is implemented here as a set of modules. For convenience the algorithm (and associated modules) will be hereafter referred to collectively as Barnthom. Complementing these is a matching module based on Hannah's algorithm [7] (see section 8), which takes the original images and performs the feature extraction and feature matching in the one module (due to difficulties in separating the two phases for this particular algorithm). The Hannah module works with the generic image format and creates a generic flow field, both of which are described in Appendix A.

7 BARNTHOM RELAXATION MATCHING MODULES

The Barnthom modules are variants of the weight initialisation and probability relaxation algorithm described by Barnard and Thompson. These implementations differ from the description in the original paper [3], in that they are not constrained to any one feature type, the initialisation stage has been separated out from the relaxation stage, and various different methods of generating initial matches and neighbours have been used. The modules are also capable of accepting feedback in the form of local or general flow information, which may be used to more accurately specify search areas.

7.1 Description of the Barnthom Relaxation Matching Algorithm

The Barnthom relaxation matching algorithm requires two feature fields of the same dimension as inputs. The basic method involved is to make some initial estimate of the probability of a given feature in the first image matching each feature in the second image which lies within same predefined region in the second image. Calculated with this initial probability is a disparity which measures the difference in location (with respect to the image coordinates) of the two features.

Once the initial probabilities have been calculated the relaxation phase is entered. The relaxation scheme is conceptually described as follows. For each potential disparity for each feature in the first image, find out how many of its neighbours (in a local region about the primary image point) have potential matches with a similar disparity. If a large number of neighbours do have a similar disparity then this potential match is more likely to be the correct match. If only a very small number of the feature's neighbours have a similar disparity then the probability of this

potential match being the correct match decreases. This relaxation process is performed iteratively a specified number of times, hopefully often enough to settle the disparity field into a stable (and correct) pattern.

A more formal description of the algorithm can be found in Barnard and Thompson [3] or Hildebrandt [8].

The implementation of this algorithm allows for a number of different weight initialisation schemes to be chosen, and also a number of different relaxation schemes, each with its own module. A naming convention (detailed below) has been adopted to uniquely identify each different combination.

7.1.1 Weight Initialisation Stage

The four methods for weight initialisation consist of two methods where feature similarity is based upon the sum of squares of differences over a five by five mask applied to the initial grey level image, and two methods for which feature similarity is based on the squared Euclidean distance between the features as described in feature space (in other words, the sum of the squares of differences between the elements of the two feature vectors). Within each of these two major categories of weight initialisation are modules which use either a uniform specified window size in pixels for choosing the valid possible matches, or a uniform number of possible matches, choosing the specified number of closest matches as the valid possible matches.

In order to differentiate the four methods of weight initialisation a module naming convention has been adopted. The four initialisation modules are called Init_Relax(focus_corr), Init_Relax(focus_gen), Init_Relax(NN_corr) and Init_Relax(NN_gen). The substrings "NN" and "focus" indicate respectively whether a specified number of closest matches or a specified search window size are used to determine the valid possible matches. Substrings "corr" and "gen" indicate respectively whether feature similarity is based on a comparison of windowed areas on the initial grey level images or on the squared Euclidian distance in feature space.

All four modules have the capability of accepting a flow field, where the flow vectors give a local estimate of the centre of the search area for possible matches. If no flow information is available then six parameters provide a two dimensional flow estimate within the image, which is then used to estimate the centre point for searching.

7.1.2 Relaxation Stage

The four methods for relaxation consist of two modules for which the support neighbourhood is defined as a constant sized window (specified in pixels), and two modules where the support neighbourhood is defined as a specified

number of closest neighbouring features. Within each of these two major categories there is a module which applies a uniform support weighting to each neighbour in the neighbourhood, and a module which applies a gaussian support weighting, based on the distance from the feature in terms of pixels or number of closer neighbours, which ever is relevant.

The four modules in this stage are named Relaxation, Relaxation_NN, Wgtd_Relax and Wgtd_Relax_NN. The "NN" substring again denotes that the support neighbourhood is defined via a specified number of closest neighbouring features rather than a specified window size. The "Wgtd" substring indicates Gaussian support weighting.

7.2 Implementation in AVS

7.2.1 General

Refer to subsection 2.1 for general information on source code, operating system, file systems, data structures and AVS module structure.

7.2.2 Source Code and File Systems

The source files together with a Makefile are contained in the directory *ii3:/gig3c/users/avs/modules_src/relaxation* on the ITD network. The relevant files are:

```
relaxation.h
Initialize/
      init_relax_conversion.c
      NN_corr/
            Makefile
            avs-init_relaxation_NN_corr.c
            init_relaxation_conversion.c
            init_relaxation_NN_corr.c
            relaxation.h
      NN_gen/
            Makefile
            avs-init_relaxation_NN_gen.c
            init_relaxation_conversion.c
            init_relaxation_NN_gen.c
            relaxation.h
      focus_corr/
            Makefile
            avs-init_relaxation_focus_corr.c
            init_relaxation_conversion.c
            init_relaxation_focus_corr.c
            relaxation.h
      focus_gen/
```

```
                Makefile
                avs-init_relaxation_focus_gen.c
                init_relaxation_conversion.c
                init_relaxation_focus_gen.c
                relaxation.h
relaxation/
        relaxation_conversion.c
        relaxation.h
        Barnthom_Relax/
                Makefile
                avs_relaxation.c
                relaxation.c
        NNRelax/
                Makefile
                avs_relaxation.c
                relaxation.c
        Weighted_BT_Relax/
                Makefile
                avs_Weighted_relaxation.c
                Weighted_relaxation.c
        Weighted_NN_Relax/
                Makefile
                avs_weighted_nn_relaxation.c
                weighted_nn_relaxation.c
```

The binary executables are located in the directory *ii3:/gig3c/users/avs/modules_bin*:

avs_init_relaxation_NN_corr
avs_init_relaxation_NN_gen
avs_init_relaxation_focus_corr
avs_init_relaxation_focus_gen
avs_relaxation
avs_relaxation_NN
avs_Weighted_relaxation
avs_weighted_relaxation_NN

Help files (text) are located in the directory *ii3:/gig3c/users/avs/help*:

```
Init_Relaxation_NN_corr.txt
Init_Relaxation_NN_gen.txt
Init_Relaxation_focus_corr.txt
Init_Relaxation_focus_gen.txt
Relaxation.txt
Relaxation_NN.txt
Weighted_Relax.txt
Weighted_Relax_NN.txt
```

7.2.3 Data Structures

7.2.3.1 AVS

The AVS field data structure is used to pass both the incoming feature fields and the outgoing flow fields. The feature fields are effectively two dimensional arrays with each element of the array containing an arbitrary length vector (defined for each feature type). The flow field is also a two dimensional AVS field, where each element contains four values (primary image, secondary image, vertical displacement between features, horizontal displacement between features). These data structures are described in more detail in Appendix A.

7.2.3.2 Other

Within the modules there are a number of data structures used to contain information. There are three dimensional arrays to handle working copies of the input and output fields as well as dynamically allocated arrays to handle the intermediate data. The following data structures and their logical interpretation are important:

7.2.3.3 Initialisation Stage

declaration: **float ***Feat1, ***Feat2;**
access: **Feat1[i][j][l] = *(*(*(Feat1+i)+j)+k);**
These are dynamically allocated three dimensional arrays which hold copies of the input feature fields. The first two dimensions correspond to pixel location, while the third index accesses the feature vector. The first element of the third dimension holds the initial grey image, the second and further elements hold the actual feature parameters. See also Appendix A.

declaration: **float ***Flow_In;**
access: **Flow_In[i][j][l] = *(*(*(Flow_In+i)+j)+l);**
This is exactly a representation of a flow field as defined in Appendix A.

declaration: **float ***Prob_Table;**
access: **Prob_Table[f1][f2][l] = *(*(*(Prob_Table+f1)+f2)+l);**
This dynamically allocated three dimensional array will hold the probability initialisation for the later relaxation process. The first dimension is accessed by the feature labels in the primary image. The second dimension is accessed by the match number of the possible match of the feature in the primary image to the feature(s) in the secondary image. The third dimension is three

elements long and holds the horizontal disparity between the primary feature and its possible match, the vertical disparity between the feature and its match, and the initial estimate of the probability of that match being the actual match. The first two positions in the second dimension are reserved for other information. The first position (Prob_Table[f1][0][l]) holds the horizontal location of the primary feature, the vertical location of the primary feature and the number of possible matches for this feature (i.e. the length of the second dimension). The second position (Prob_Table[f1][1][l]) holds the probability of not having a valid match.

```
struct List_Node {
    int     dX;
    int     dY;
    float   Prob_Match;
    struct  List_Node     *Next_Node;
    };
```

This structure is used to set up temporary linked lists to hold the possible matches for each primary feature. This list is copied to the Prob_Table. When Nearest Number of matches is the chosen technique, this list is used with in-order insertion to quickly find the specified number of closest matches.

7.2.3.4 Relaxation Stage

declaration: `float ***Prob_Table;`
access: `Prob_Table[f1][f2][l] = *(*(*(Prob_Table+f1)+f2)+l);`

This dynamically allocated three dimensional array will hold the probability initialisation for the later relaxation process, and also holds the updated probabilities throughout the relaxation. The first dimension is accessed by the feature labels in the primary image. The second dimension is accessed by the match number of the possible match of the feature in the primary image to the feature(s) in the secondary image. The third dimension is three elements long and holds the horizontal disparity between the primary feature and its possible match, the vertical disparity between the feature and its match, and the initial estimate of the probability of that match being the actual match. The first two positions in the second dimension are reserved for other information. The first position (Prob_Table[f1][0][l]) holds the horizontal location of the primary feature, the vertical location of the primary feature and the number of possible matches for this feature (i.e. the length of the second dimension). The second position (Prob_Table[f1][1][l]) holds the probability of not having

a valid match most of the time, but holds the most likely match information at the end of the module.

declaration: float **Prob_Temp;
access: Prob_Temp[f1][f2] = *(*(Prob_Temp+f1)+f2);
This array is the same as Prob_Table, except that it only holds the probability of match information. It is used as temporary storage of updated probabilities before being copied back into Prob_Table at the end of each iteration.

```
struct NeighNode {
    int     Neighbour;
    struct NeighNode    *Next;
    } **Neighbourhood;
```
access: *Neighbourhood[f1];
This array of pointers to structures is an array of linked lists, accessed by the feature of interest, which holds the neighbours of the feature of interest (as an index into Prob_Table).

7.2.4 *Function Descriptions*

As in previous sections, the following descriptions are not intended to be an exhaustive documentation of functional interactions and dependencies, rather they should be used as a guide to understanding and maintaining the module.

Figure 8 gives an hierarchical description of the functions and their dependencies for the four initialisation modules. The function names may vary slightly from those specified in the descriptions that follow.

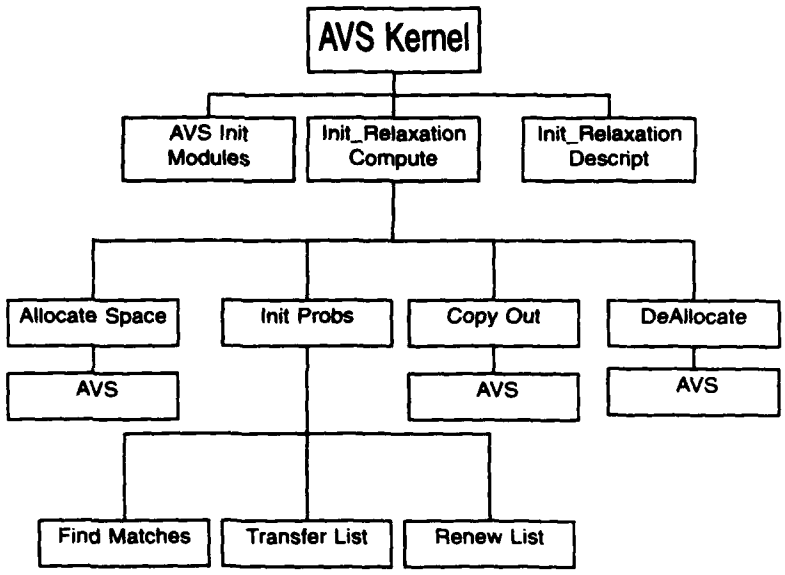


Figure 8 : Hierarchical Functional Block Diagram for Initialisation

Init_Relaxation_compute

source code file:	avs_init_relaxation_<focus/NN>_<corr/gen>.c
called by:	AVS kernel
calls to:	Allocate_Space
	Init_Probs
	Copy_Out
	DeAllocate

Description:

Init_Relaxation_compute is the computation function (refer subsection 2.1.5) for the AVS module and is the entry point for the user supplied code.

Allocate_Space

source code file:	init_relax_conversion.c
called by:	Init_Relaxation_compute
calls to:	AVS

Description:

This routine allocates space for and copies the original images to the working data structures.

Init_Probs

source code file:	init_relaxation_<focus/NN>_<corr/gen>.c
called by:	Init_Relaxation_compute
calls to:	Find_Matches
	Transfer_List
	Renew_List

Description:

This routine loops through all of the primary features finding the possible set of matches for each feature (with calls to Find_Matches). It then calls Transfer_List which calculates the initial probabilities and writes these values to the array data structure.

Find_Matches

source code file:	init_relaxation_<focus/NN>_<corr/gen>.c
called by:	Init_Probs
calls to:	None

Description:

This function scans through the secondary image looking for any features which may be considered to be valid matches to the primary feature in question. The actual method for determining validity depends on whether it is a "focus" or a "NN" initialisation module. The result is a linked list of valid match points.

Transfer_List

source code file: init_relaxation_<focus/NN>_<corr/gen>.c
called by: Init_Probs
calls to: None

Description:

This function takes the linked list of valid match points found in "Transfer_List" and for each match calculates the initial probability of match (the method being dependent on whether the module is a "corr" or a "gen" type initialisation module). These match points are then written into the Prob_Table array structure used for passing the information.

Renew_List

source code file: init_relaxation_<focus/NN>_<corr/gen>.c
called by: Init_Probs
calls to: None
Description:

This function removes and frees the elements of the linked list found in "Find_Matches".

Copy_Out

source code file: init_relax_conversion.c
called by: Init_Relaxation_compute
calls to: AVS

Description:

This function creates space for the output data structures and copies the information calculated in "Init_Probs" (Prob_Table) to the output.

DeAllocate

source code file: init_relax_conversion.c
called by: Init_Relaxation_compute
calls to: AVS

Description:

This routine cleans up the intermediate data structures and frees up any dynamically allocated memory.

Figure 9 gives an hierarchical description of the functions and their dependencies for the four relaxation modules. The function names may vary slightly from those specified in the descriptions that follow.

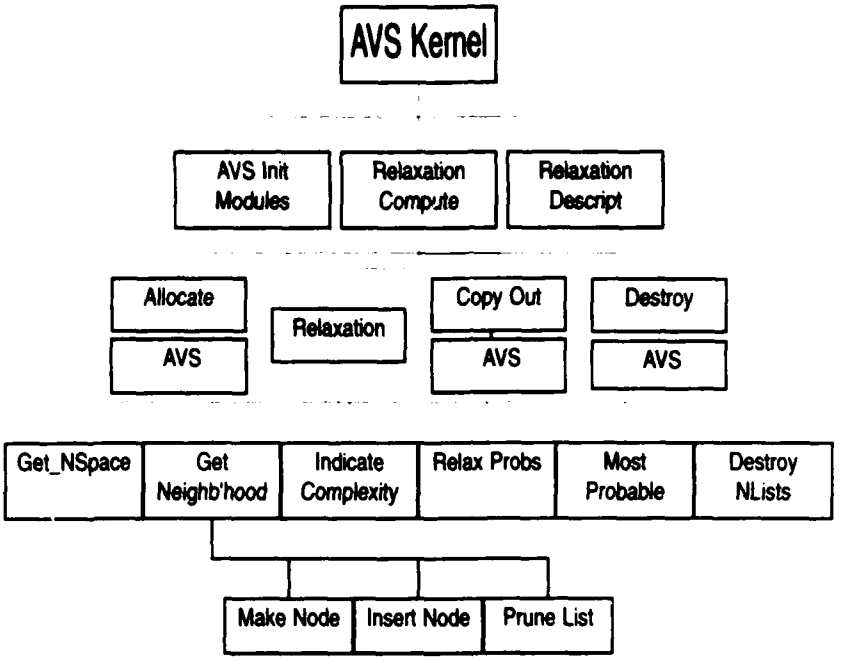


Figure 9 : Hierarchical Functional Block Diagram for Relaxation

Relaxation_compute

source code file: avs_relaxation.c, avs_relaxation_NN, weighted_relaxation.c, weighted_relaxation_NN.c

called by: AVS kernel

calls to: Allocate_Space
Relaxation
Copy_Out
Destroy

Description:

"Init_Relaxation_compute" is the computation function (refer subsection 2.1.5) for the AVS module and is the entry point for the user supplied code.

Allocate_Space

source code file: relaxation_conversion.c
called by: Relaxation_compute
calls to: AVS

Description:

This routine allocates space for and copies the original information to the working data structures.

Relaxation

source code file: relaxation.c, relaxation_NN.c
Wgtd_relaxation.c, Wgtd_relaxation_NN.c
called by: Relaxation_compute
calls to: Get_NSpace
Get_Neighbourhood
Indicate_Complexity
Relax_Probs
Most_Probable
Destroy_NLists

Description:

This routine is the main entry point to the actual working code, and is used basically to make the appropriate calls to the various functions.

Get_NSpace

source code file: relaxation.c, relaxation_NN.c
Wgtd_relaxation.c, Wgtd_relaxation_NN.c
called by: Relaxation
calls to: None

Description:

This routine creates space for the neighbourhoods.

Get_Neighbourhood

source code file: relaxation.c, relaxation_NN.c
Wgtd_relaxation.c, Wgtd_relaxation_NN.c
called by: Relaxation
calls to: Make_Node
Insert_Node
Prune_List

Description:

This routine finds the set of neighbouring features to each feature and stores them in the neighbourhood array. The specific algorithm depends on which module is being used.

Indicate_Complexity

source code file: relaxation.c, relaxation_NN.c
Wgtd_relaxation.c, Wgtd_relaxation_NN.c
called by: Relaxation
calls to: None

Description:

This routine measures the complexity of the algorithm and estimates how long the module will take to complete (and depends on arbitrary measurements for a single SPARC 10 computer with an unknown environment).

Relax_Probs

source code file: relaxation.c, relaxation_NN.c
Wgtd_relaxation.c, Wgtd_relaxation_NN.c
called by: Relaxation
calls to: None

Description:

This routine performs a single update and normalise step using the neighbourhood relaxation described by Barnard and Thompson [3]. Variations include some weighting on the support dependent upon distance from the central feature of interest.

Most_Probable

source code file: relaxation.c, relaxation_NN.c
Wgtd_relaxation.c, Wgtd_relaxation_NN.c
called by: Relaxation
calls to: None

Description:

This routine places the most probable match point into the second position of the Prob_Table, for later extraction and insertion into the output flow field.

Destroy_NLists

source code file: relaxation.c, relaxation_NN.c
Wgtd_relaxation.c, Wgtd_relaxation_NN.c
called by: Relaxation
calls to: None

Description:

This routine frees the space used in the neighbourhood lists.

Copy_Out

source code file: relaxation_conversion.c
called by: Relaxation_compute
calls to: AVS

Description:

This function creates space for the output data structures and copies the information calculated in "Init_Probs" (Prob_Table) to the output.

Destroy

source code file: relaxation_conversion.c
called by: Relaxation_compute
calls to: AVS

Description:

This routine cleans up the intermediate data structures and frees memory.

7.3 Example Network and Output

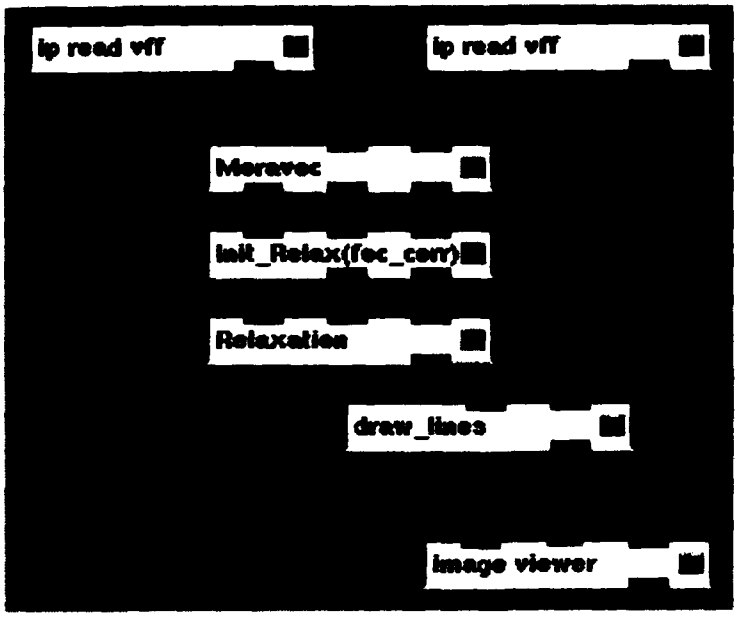


Figure 10: Example AVS Network for Relaxation Matching

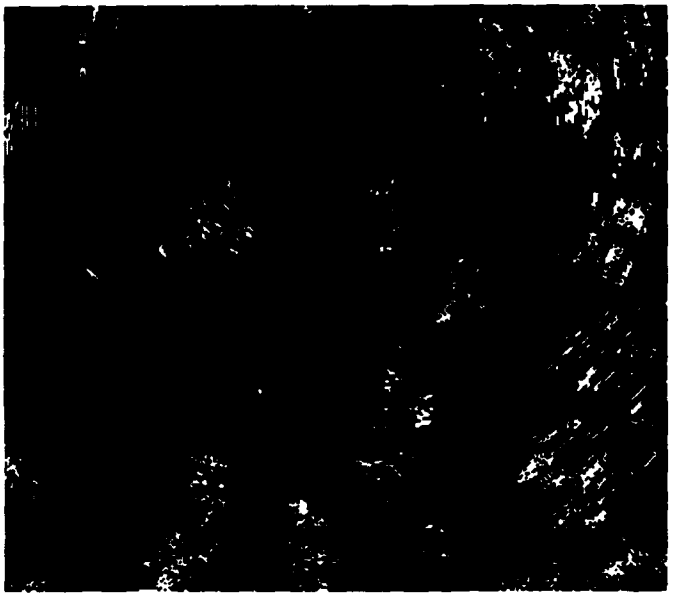


Figure 11 : Example Output for Relaxation Matching. Disparity vectors are represented by lines overlaid onto original left input image.

7.4 Comments on Parameter Values and Performance

Each of the four initialisation modules has six user selectable parameters, while the four relaxation modules each have two. A complete list and detailed explanation of these parameters appears in Appendix B.

To get reasonable performance takes about 30 minutes for 512*512 images on a Sun SPARCstation 10. In general, the longer the run time the better the performance, although recognisable match flow can be found in as little as five minutes. The "NN" methods (both Init_Relaxation and Relaxation) seem to give slightly quicker and possibly more accurate results than the "focus" methods, but this depends somewhat on the image characteristics.

Module performance is very much dependant on the values selected for the parameters

Init_Relaxation
In general it is not desirable to have more than about 1000 features although this can be any number if the user is prepared to wait. The search-window size should be JUST large enough to account for the largest estimated flow: the bigger the window the longer the execution time (with a roughly quartic relationship). A window size of 8 pixels (focus) in each direction is a good starting point. A window size of 20 features (NN) is also a reasonable starting point. Obviously, some prior estimate of flow will reduce the time by allowing much smaller search windows to be used.

Relaxation
Again the same trade-off exists for window size vs run time (although the relationship is now quadratic). Reasonable window sizes are a bit larger: 15 pixels (focus) or 25 features (NN). Ten iterations are usually plenty. These values will result in a run time of 5-10 minutes.

A qualitative estimate of performance is as follows:

Time (min)	Number of Matches	% Accuracy (Self Consistency)
5	400	%60
30	600	%70
180	800	%75-85 ????

7.5 Future Directions

The next stage for these modules is to test them to determine optimal parameters for particular image types. Perhaps an algorithm could be developed to determine these parameters during execution. A similar algorithm based on Kittler, Christmas and Petrou's method [10] is currently being developed which promises to provide a more rigorous method of determining the optimal values of the parameters involved.

8 HANNAH MATCHING MODULE

8.1 Description of the Hannah Matching Technique

This subsection outlines the system developed by Hannah [7] for the automatic matching of points in stereo digital imagery. The system employs area-based correlation as the matching mechanism, coupled with a variety of hierarchical search techniques. The techniques incorporate iterative refinement and use a best-first strategy in the matching process. Hierarchical back-matching is used to further check reliability. Matching on image patches is performed by computing the cross-correlation, normalised by mean and variance over windows surrounding the interest points. The system employs a number of different matching algorithms in succession; moving from global, relatively conservative search algorithms operating on a few points that are highly likely to be matchable, to successively more constrained searches (utilising information from preceding matches) on less promising points. Each of the algorithms makes at least some use of image pyramids (hierarchies). Two pyramids are used, one for each of the two original input images.

The algorithms were designed for use with images of highly textured natural terrain (i.e. aerial photos) rather than cultural objects with "mostly linear or ambiguous features". Nevertheless the system still rated very highly in the Image Matching Test A for ISPRIS Working Group III/4 [6], which used a wide variety of image subjects, scales and quality. Hannah claims that for the most part, the results appeared to be "reasonably" correct, even for large disparity ranges within small areas of an image, and that the system is robust and is widely applicable.

The main stages of the complete matching system as described by Hannah are outlined in subsections 8.1.1 through 8.1.5. Aspects of our implementation are detailed in subsection 8.2. Note that at present, primarily due to lack of programming resources, only the first two stages have been implemented. Hopefully further stages will be implemented in the not too distant future.

8.1.1 Selection of Interest Points

As in Moravec feature extraction, an interest operator is first passed over one of the input images, with local peaks in the output designated as "interesting points". Hannah proposed an operator that is a "product of the image variance and the minimum of ratios of directed differences over windows of a specified size", designed to penalise windows with low information and windows where the information is contained in strongly linear edges. The image is then subdivided into a grid of subimages, and the relative (interest operator output) strengths of the best few interest points within each subimage recorded. This is meant to provide a good spread of interest points throughout the original image.

8.1.2 Unconstrained Hierarchical Matching

Image pyramids are first created for each of the two input images (hereafter designated as image 1 and image 2). Each pyramid comprises a number of successively lower resolution (smaller) images, with each image produced by convolving the image below it with a Gaussian and subsampling by a factor of two. The topmost image in the pyramid is approximately the size of the correlation window.

The unconstrained hierarchical matching technique starts with a point in image 1 (usually the most interesting point in each subimage) and traces it back up through the image hierarchy (by scaling down the point's coordinates) until it reaches the topmost level. A window is then taken about the point and a search is made to find a correlation match with a similarly windowed area on the corresponding image in the image 2 hierarchy. The search commences with a 2D spiral search, followed by a hill-climbing search for the maximum correlation between the image windows. The match point is then moved down the image hierarchy and becomes the starting point for a hill-climbing search at the next level. As the absolute size of the correlation window does not change between levels, the relative size of the window with respect to the image at each level in the hierarchy increases as the hierarchy is traversed upwards. This means that matches are effectively performed first over the entire image, then over increasingly local areas, hence allowing overall image structure to set the context for a match.

At any level in the hierarchy, matches with poor correlation (against an absolute or auto correlation based threshold) are discarded. Each match must also be confirmed by back matching: i.e. having found a match between point p_1 in image 1 and p_2 in image 2, the point p_2 is subjected to the same hierarchical matching algorithm to arrive at a match p_1' in image 1. If p_1 and p_1' differ by more than one pixel the entire match is discarded.

8.1.3 Camera Modelling

The remaining search algorithms make use of epipolar constraints, thus requiring some knowledge of camera geometry. If camera models are unavailable or unreliable the system can calculate a simplistic relative camera model from the disparity information produced by the first matching stage above. This is done by searching for five parameters that describe the relative positions and orientations of two ideal pinhole cameras. The object of the search is to minimise the error between p_2 in image 2 and the epipolar line produced when p_1 in image 1 is projected into space, then into image 2 through the hypothesised pinhole cameras. The resulting equations and their analytic derivatives are linearised, and an iterative process is carried

out so as to cull unreliable points until convergence to a consistent model is obtained (otherwise failure is reported).

8.1.4 Epipolar Constrained Hierarchical Matching

This stage is very similar to unconstrained hierarchical matching, but reduces the search for matching points to one dimension (within a specified distance of the appropriate epipolar line). It is applied to any of the two most interesting points for each subimage which were not already matched during the unconstrained hierarchical matching stage.

8.1.5 Anchored Matching

Using reliable matches produced from earlier stages as "anchor points", and *assuming a continuous disparity field*, a very local search is carried out for each candidate match point using the epipolar constraint together with the requirement that its disparity must lie within the disparity interval of anchor points in the current and neighbouring subimages. Context for the match may also be obtained by moving one or two levels up the image hierarchy. All matches must pass the same tests as for the first two stages outlined above, as well as an anchored-matching version of the back-matching test.

8.2 Implementation in AVS

8.2.1 General

At present only the first two stages (interest point selection and unconstrained hierarchical matching) of the Hannah matching technique have been implemented in AVS. It is proposed to add further stages in the future (see subsection 8.4) as programming support becomes available.

As outlined in section 2, the matching routine is implemented as a non-AVS-specific "core" function that interfaces to the AVS environment via an AVS module "wrapper". Refer to subsection 2.1 for details on general implementation aspects common to all the modules documented in this report.

The remainder of subsection 8.2 provides details on the implementation of the Hannah routine. Comments on the source code and file system structure (subsection 8.2.2) are followed by a brief description of the important data structures (8.2.3) and finally a description of the most important functions and their role within the function hierarchy (8.2.4). Note that this subsection does not specifically deal with the AVS interface (AVS user documentation for the module is provided in Appendix B). For illustrative purposes, however, Figure 12 shows typical AVS network interconnections for the module while Figure 13 shows an example user interface panel (easily reconfigurable under the AVS layout editor).

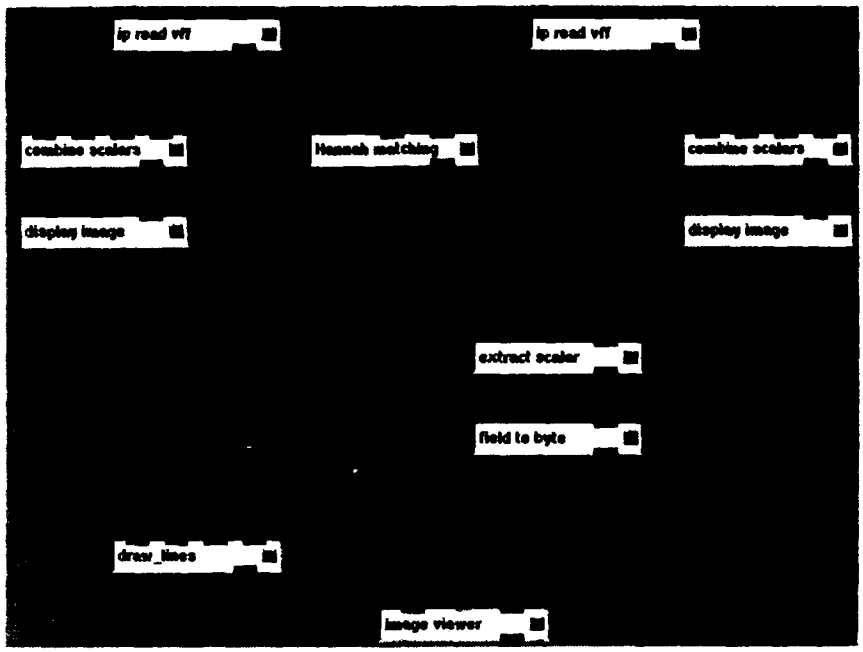


Figure 12 : Example AVS Network Incorporating Hannah Matching Module

Figure 13 : Example Hannah User Interface with Output Display. User selectable parameters for the algorithm are at left. Image Viewer shows output as disparity vectors overlaid onto original left hand input image.

8.2.2 Source Code and File Systems

Source files together with a Makefile are contained in directory *ii3:/gig3c/users/avs/modules_src* on the server *ii3* on the ITD network. The relevant files are :

```
Makefile.hannah_avsmodule
hannah_avsmodule.c
hannah.h
hannah.c
create_image_hierarchies.c
create_intpoint_hierarchies.c
uhm.c
arrayalloc.c
```

The binary executable *hannah_avsmodule* is located in directory */gig3c/users/avs/modules_bin*.

8.2.3 Data Structures

8.2.3.1 AVS

The AVS *field* data structure is used to pass image and disparity information into and out of the Hannah matching module. During the development phase it has been convenient to store module output as an AVS 2D field of floating point 4-vectors that contains both the original images together with x and y disparity information (see the descriptions in section 2 and Appendix A). The relative sparseness of this disparity information means this is a memory-inefficient representation, however in the authors' opinion this disadvantage is outweighed by the convenience of using relatively straightforward 2D array indexing and standard AVS field access macros in implementing search techniques during matching. Nevertheless the benefits and trade-offs involved in using a structure such as an AVS 1D n-vector field for conveying generic feature lists between modules warrants further investigation.

8.2.3.2 Other

Within the non-AVS-specific code there are two specially created C structures which merit some explanation. Both are defined in the header file `hannah.h`.

```
struct hierImage{
int   hier_level;
int   xsize, ysize;
float         **im;
struct hierImage  *parent, *child;
} hierImage;
```

This is the template for the image pyramid structure. It is a doubly linked list, with fields recording the image hierarchy level, image dimensions, and a pointer to the image data.

```
struct        hierIntPoint{
int   hier_level;
int   x, y;
int   xmatchdisp, ymatchdisp;
float         corr;
int   status;
struct hierIntPoint    *prev, *next;
} hierIntPoint;
```

This is also a doubly linked list, with each node containing information on a particular interest point at a particular level in the image hierarchy. Fields record the point's coordinates, the *x* (vertical) and *y* (horizontal) displacements to the matched point (if one exists), the correlation value of the matched point, and match status (BACK_MATCHED or NOT_MATCHED). Note that the forward and backward links in the list are between the same interest point at different levels in the image hierarchy. The intpoint array (used in functions *hannah*, *create_intpoint_hierarchies* and *uhm*) is a 3D array of these linked lists. The array indices reference the subimage cell location and the "order of merit" of the interest point within the subimage (e.g. intpoint[0][0][3] is the 4th highest ranking interest point in the top left subimage) and the array fields contain associated properties (e.g. *intpoint[0][0][3].x* and *intpoint[0][0][3].y* give the *x* and *y* positions of the interest point at the particular image hierarchy level specified in intpoint[0][0][3].hier_level). It is linked to other levels via intpoint[0][0][3].prev and intpoint[0][0][3].next. In this way the location of each interest point can be tracked up and down the image hierarchy.

8.2.4 Function Descriptions

This subsection describes the more important C functions making up the module, together with additional comments on certain aspects of the implementation. It is not intended to be exhaustive, but rather is presented

as an aid in following the flow of processing and to clarify some areas that may be confusing. Figure 14 depicts the hierarchy of function calls throughout the routine.

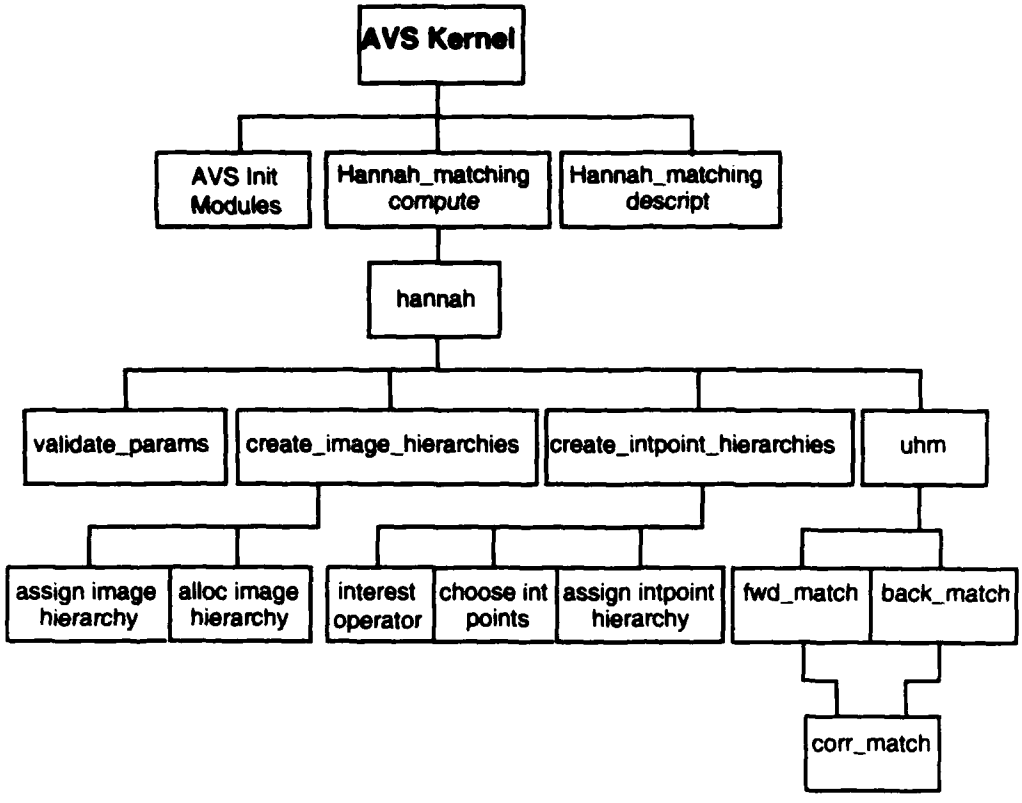


Figure 14 : Function Hierarchy for Hannah Matching

Hannah_matching_compute

source code file: hannah_avsmodule.c
called by: AVS kernel
calls to: hannah

Description:

Hannah_matching_compute is the computation function (refer subsection 2.1.5) for the Hannah matching module. It receives the user inputs from the AVS interface, copies image information from AVS-specific data structures into standa.d C arrays and structures, and passes them to *hannah*. In this way, *hannah_avsmodule* acts as an AVS wrapper to the core hannah code. The left and right input images are converted to floating point and copied to the output field's first and second vector elements, while the disparity information calculated by *subdirectory* is copied from the disparity[][][][] array into the third and fourth vector elements.

hannah

source code file:	hannah.c
called by:	hannah_avsmodule
calls to:	validate_params
	create_image_hierarchies
	create_intpoint_hierarchies
	uhm

Description:

This is the main function for the Hannah algorithm. It consists principally of a series of function calls (described below), followed by a triple loop that copies disparity information for all back-matched points from the appropriate linked list fields into the disparity array. Indices are arranged such that disparity[0][j][i] and disparity[1][j][i] store respectively the *y* and *x* direction disparities for the interest point at coordinates *(i,j)*.

validate_params

source code file:	hannah.c
called by:	hannah
calls to:	

Description:

This function performs simple checks on the values of some of the input parameters. The macros MAXSUBX, MAXSUBY and MAXINTPERCELL are defined in the header file `hannah.h`.

create_image_hierarchies

source code file:	create_image_hierarchies.c
called by:	hannah
calls to:	alloc_image_hierarchy
	assign_image_hierarchy

Description:

This function takes the two input images, contained in arrays *im1* and *im2*, and creates a linked list implementing an image hierarchy. The function *alloc_image_hierarchy* allocates the memory for the image hierarchy structure, while *assign_image_hierarchy* take the original input image and successively smooths (3x3 Gaussian operator) and subsamples (by a factor of 2 each time) to create the hierarchy data. The subsampling process ends when an image is obtained that is unable to be subsampled any further without becoming smaller than the size of the (user-selectable) correlation window size. Upon exit, *assign_image_hierarchy* sets the argument *root* to the address of the hierImage linked list node relating to the smallest image in

the hierarchy, while the argument *no_children* stores the number of child images in the hierarchy.

create_intpoint_hierarchies

source code file: create_intpoint_hierarchies.c
called by: hannah
calls to: interest_operator
choose_int_pts
assign_intpoint_hierarchy

Description:

This function first calculates an interest value for each image pixel in input image 1. This is carried out by the function *interest_operator* which passes a 7 x 7 interest operator over the image. The form of the operator is not explicitly provided in [7], but has been interpreted from its description to be

$$O(I(x,y)) = \sum_{i=-3}^{3}\sum_{i=-3}^{3}\left(I(x+i,y+j)-\bar{I}\right)^2 * \min\left(r, r^{-1}, s, s^{-1}\right) \quad (8.1)$$

where $$r = \left|\frac{\sum_{i=-3}^{3}\sum_{j=-2}^{3}\left(I(x+i,y+j)-I(x+i,y+j-1)\right)}{\sum_{i=-2}^{3}\sum_{j=-3}^{3}\left(I(x+i,y+j)-I(x+i-1,y+j)\right)}\right|$$ and

$$s = \left|\frac{\sum_{i=-3}^{2}\sum_{j=-2}^{3}\left(I(x+i,y+j)-I(x+i+1,y+j-1)\right)}{\sum_{i=-2}^{3}\sum_{j=-2}^{3}\left(I(x+i,y+j)-I(x+i-1,y+j-1)\right)}\right|$$

and where summations are performed over the 7 x 7 window centred on the pixel with coordinates (x,y). Again $I(x,y)$ specifies the pixel value at the location (x,y), while $\bar{I}$ is the mean pixel value in the window. The operator does not need to be individually calculated in full at each pixel due to the redundancies that arise from overlapping as the operator window moves around the image. Calculation time is reduced by the storing and updating of a variety of running totals.

A more appropriate interpretation of the operator, however, may be

$$O(I(x,y)) = \sum_{i=-3}^{3}\sum_{i=-3}^{3}\left(I(x+i,y+j)-\bar{I}\right)^2 * \min\left(c, c^{-1}, d, d^{-1}\right) \quad (8.2)$$

where $c = \dfrac{\sum_{i=-3}^{3}\sum_{j=-2}^{3}|I(x+i,y+j)-I(x+i,y+j-1)|^2}{\sum_{i=-2}^{3}\sum_{j=-3}^{3}|I(x+i,y+j)-I(x+i-1,y+j)|^2}$ and

$$d = \frac{\sum_{i=-3}^{2}\sum_{j=-2}^{3}|I(x+i,y+j)-I(x+i+1,y+j-1)|^2}{\sum_{i=-2}^{3}\sum_{j=-2}^{3}|I(x+i,y+j)-I(x+i-1,y+j-1)|^2}$$

This version makes use of the total energy of edges in each direction across all pixels in the window, instead of the average edge strength across the window.

Once the interest $O(I(x,y))$ values have been computed, the original input images are divided up into a regular grid of subimages, with grid spacings determined by the user-selectable variables *nsubx* and *nsuby*. Function *choose_int_pts* chooses the best <int_per_cell> interest points in each subimage in order of merit, and these are then entered by the function *assign_intpoint_hierarchy* into intpoint[][][], an array of (hierarchically) linked interest points.

uhm

source code file:	uhm.c
called by:	hannah
calls to:	fwd_match
	back_match

Description:

This function performs unconstrained hierarchical matching on the selected interest points. The function *fwd_match* is first called: this carries out a hill climbing correlation search for the best matching point at each level in the image 2 hierarchy, starting each time from the point matched at the previous image hierarchy level. The hill climbing search proceeds by moving to the highest (correlation) valued immediate neighbour of the current image 2 search point, and ends when correlation is a local peak. The correlation calculation and hill-climbing technique are actually carried out in the function *corr_match*. Correlation is calculated via

$$R(w_1,w_2) = \frac{\sum_{l=1}^{N}\sum_{m=1}^{N} w_1(l,m)w_2(l,m)}{\left(\sum_{l=1}^{N}\sum_{m=1}^{N} w_1^2(l,m)\right)^{\frac{1}{2}}\left(\sum_{l=1}^{N}\sum_{ml=1}^{N} w_2^2(l,m)\right)^{\frac{1}{2}}} \quad (8.3)$$

where w_1 and w_2 are $N \times N$ windows located in image 1 and image 2 respectively (the window size N is user selectable via *win_size*). The x and y displacements for the match at each level in the image hierarchy are recorded in intpoint[][][].xmatchdisp and intpoint[][][].ymatchdisp, and the correlation is recorded in intpoint[][][].corr. The *back_match* function then carries out back matching for each matched point in image 2, again in an hierarchical fashion with the *corr_match* function reused to carry out the hill-climbing correlation matching at each level. Results are then compared to the locations of the original interest points in image 1. If the absolute values of both the x and y differences are not greater than one pixel the back match is deemed to have confirmed the forward match and the status element of the intpoint is set to BACK_MATCHED (otherwise it is set to NOT_MATCHED).

8.3 Comments on Parameter Values and Performance

There are five user selectable parameters associated with this module (a list and detailed explanation of these appears in Appendix B). To obtain a reasonable set of matches without an overly long wait, values of 8 for each of **num x subimages** and **num y subimages**, 11 for **window size**, and 5 each for **int points per cell** and **num uhm points** are a good starting point. These settings will require about 30 seconds run time and produce around 200 matches for a 512*512 image.

Although rigorous testing of the algorithm on a range of image types and noise levels is still to be carried out, on a qualitative level (based on a visual inspection of the two input images and the consistency of the generated disparity flow field) performance is generally very good, with success rates of around 90 - 95% over a wide range of parameter settings. Run time naturally increases with image size, the number of matches that the user wishes to obtain (i.e. with a larger number of subimages and points selected per subimage) and the size of the correlation window used in the matching.

Rough indicators of run times are as follows (512*512 image on Sun SPARCstation 10):

Time (min)	Matches
0.5	200
0.7	300
2.5	1100

8.4 Future Developments

Only the first two stages (see subsections 8.1.1 and 8.1.2) of Hannah's complete algorithm have been implemented to date. While these already provide quite

reasonable results (on a limited set of test images) at a qualitative level, implementing the remaining three stages (camera modelling, epipolar unconstrained hierarchical matching and anchored matching) should further improve performance. Nevertheless quantitative testing is still required in order to properly assess performance over a variety of realistic image types and depicted objects.

The camera modelling stage of the algorithm is a likely candidate for separation into an independent module. An AVS motion estimation module based on a method developed by Toscani and Faugeras [14] has already been implemented (to be described in a future report), and its use via some sort of feedback loop should be investigated. In addition, the feature extraction and matching tasks can also be implemented as separate modules. This allows the functions to be used on a greater variety of feature detection and matching problems: for example, it is proposed to further investigate the changes in performance when interest points are identified by three different modules based on equations (8.1) and (8.2) and the Moravec interest operator (subsection 4.1). It is unlikely, however, that the matching techniques outlined in subsections 8.1.2, 8.1.4 and 8.1.5 could be implemented as generic feature matching modules as they rely strongly on the assumption that the underlying objects being matched are images (as opposed to abstract feature sets).

9 CONCLUSIONS

This report has described a number of algorithms which have been implemented in a modular and interchangeable way to perform the first two layers of a system for automatic shape reconstruction from stereo imagery. These two layers are the feature extraction layer and the feature matching layer. Within the feature extraction layer there are currently two algorithms, namely "Moravec", a point based feature extractor and "FEX" which delivers edge based features. The feature matching layer currently has "Barnthom" which is a window based relaxation scheme. Spanning both layers at the one time is the "Hannah" algorithm which performs an hierarchical region based match.

Although designed to form part of a modular stereo photogrammetry system the algorithms have been implemented in as generic a manner as possible so as to have wider applicability, e.g. for use in general low level vision and image processing applications or in data fusion systems. The matching algorithms in particular are not restricted to image data types, and are capable of being applied to any matching problem in which tokens have been identified by locations in some higher dimensional vector space.

In the near future rigorous testing of these algorithms will be carried out with calibrated data in order to evaluate their robustness in the presence of noise and their performance for differing image characteristics. At the present stage, a qualitative assessment of suitable choices of input parameter values (see appropriate sections in main body of report) and comparative algorithm performance has been made for a limited set of test images. The Barnthom relaxation algorithm tends to generate a large number of matches without

necessarily having a high success rate (around 60% - 85%). Of the modules implementing the Barnthom variants, the "NN" methods seem to give slightly quicker and possibly more accurate results than the "focus" methods (in both the initialisation and relaxation stages), but this can depend somewhat on the image characteristics. Performance can be significantly improved by reverse matching (typically pruning the number of matches by 25 - 40%). Performance should also further improve if provided with a prior estimate of disparity flow (say from Hannah) as an additional input. For raw image data however, Hannah seems more reliable and more accurate than Barnthom (probably due largely to its in-built hierarchical back matching step) with a success rate of around 90 - 95% throughout a wide range of parameter settings. The module also takes significantly less time to execute than Barnthom for a similar number of matches. The algorithm does, however, have the disadvantage (within the context of this modular system) that feature extraction and matching components cannot readily be separated and modularised, and that they implicitly assume that data types are images.

The next stage of system development will see the introduction of an additional two layers. The camera geometry/motion estimation layer will aim to provide the automatic determination of camera geometry (or equivalently object motion) if this is not known a-priori. The depth estimation layer will then use this information together with the disparity information obtained from the matching layer to estimate depth or shape. Preliminary algorithms for these layers are currently in the testing stage. Having completed the modular framework for the full system it is then planned to further populate it with algorithms available from the research literature, test each for performance on typical data sets and to automate as much as possible the selection of appropriate input parameters and combinations of algorithms for a given image type.

Longer term research may see the fusion of these stereo techniques with alternative shape inference methods, such as the "shape from shading" algorithms being developed by the University of Adelaide component of the Visual Processing research program at the CSSIP.

10 ACKNOWLEDGMENTS

The authors would like to acknowledge the assistance and help of Dr. Janet Aisbett and Dr. John Hildebrandt, who both supplied source code and advice in the implementation of these modules, and Dr. Garry Newsam for advice and guidance in the preparation of this report.

REFERENCES

1. **Advanced Visual Systems** — *AVS User's Guide*. Advanced Visual Systems Inc., Waltham, MA., Part Number 320-0011-02 Rev B, 1992.

2. **Baker, M.S. and Newsam, G.N.** — *A Computer Package for Image Registration Using Smoothing Splines*. Research Report ERL-0804-RR, Electronics Research Laboratory, DSTO, Canberra, Australia, 1993.

3. **Barnard, S.T. and Thompson, W.B.** — 'Disparity Analysis of Images", *IEEE Transactions on Pattern Analysis and Machine Intelligence* **1980**, *2* (4), 333-340.

4. Canny, J. — "A Computational Approach to Edge Detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence* **1986**, *8* (6), 679-698.

5. Etemadi, A. — "Robust Segmentation of Edge Data", code obtained by ftp from *Percentual Groupings* archive, University of Surrey, U.K., *email:atae@spva.ph.ic.ac.uk*.

6. Gülch, E. — "Results of Test on Image Matching of ISPRS WG III/4", *International Archives of Photogrammetry and Remote Sensing*, **1988**, *27-III*, 254-271.

7. Hannah, M.J. — "A System for Digital Stereo Image Matching", *Photogrammetric Engineering and Remote Sensing* **1989**, *55*(12), 1765-1770.

8. Hildebrandt, J.H. — *Edge Based Methods for Image Matching*. Divisional Paper ITD-94-08, Information Technology Division, Electronics Research Laboratory, DSTO, Canberra, Australia, 1994.

9. Kernighan, B. and Ritchie, D. — *The C Programming Language*. Prentice-Hall, Englewood Cliffs, NJ, 1978

10. Kittler, J., Christmas, W.J., and Petrou, M. — "Probabilistic Relaxation For Matching Problems In Computer Vision", in: *Proceedings of the Fourth International Conference on Computer Vision*. Berlin, Germany, IEEE Computer Society Press, 1993; pp 666-673.

11. Moravec, H.P. "Towards Automatic Visual Obstacle Avoidance", in: *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*. Cambridge, MA, IJCAI, 1977; p 584.

12. Rasure, J.R., and Young, M. "An Open Environment for Image Processing Software Development", in: *Image Processing and Interchange: Implementation and Systems*. Ronald B. Arps, William K. Pratt (Eds), Proc. SPIE 1659, 1992; pp 300-310.

13. Russ, J.C. *The Image Processing Handbook*. CRC Press, Florida, 1992; p 125.

13. Toscani, G. and Faugeras, O. "Structure and Motion from Two Noisy Perspective Views", in: *Proceedings of the 1987 IEEE International Conference on Robotics and Automation*. Raleigh, North Carolina, IEEE Computer Society Press, 1987; Vol. 1, pp 221-227.

APPENDIX A: SUMMARY OF DATA REPRESENTATIONS

This appendix contains details of the AVS data formats used for representing the input images and the feature and flow fields.

Input Data Format

AVSfield(2D, scalar, uniform, byte)

This is a single band of an image, either an 8 bit grey image or one band of a 32 bit 4-band image. Its size (width and height) is specified within the data structure describing the field.

Feature Field Data Format

AVSfield(2D, uniform, float)

This is a 2D data structure where the number of elements at each point is undefined. Again, the size (width, height, vectorlength) is specified within the data structure describing the field. By convention the first band (band 0) is a copy of the original grey level image converted to floating point values. The remaining bands hold the feature vectors as calculated at a point. Non-features have a value of zero for the feature elements. For instance Moravec places the Moravec interest operator in the second band (Band 1). FEX places three values in Bands 1, 2 and 3. There is no convention about what values are acceptable as features at this stage.

Flow Field Data Format

AVSfield(2D,4-vector,uniform,float)

This is a 2D data structure of floating point values where each element contains exactly four items. The first band (Band 0) contains the left or primary image. The second band (Band 1) contains the right or secondary image. The remaining two bands contain respectively the "y" disparity and the "x" disparity. These values can be either:

a) The actual displacement (in y or x direction according to band) between matching features, stored as a floating point real.
or b) NotMatched (defined as a large constant in "relaxation.h").
or c) Ignored (defined as a large constant in "relaxation.h").

NotMatched means there is a feature point here, but no match was found. Ignored means that there was not a feature point here.

APPENDIX B: ON-LINE USER DOCUMENTATION

This appendix consists of direct copies of the on-line user documentation (accessible via the AVS system) for the modules described in this report. A knowledge of the AVS network environment is assumed.

AVS ON-LINE HELP PAGE FOR MORAVEC FEATURE EXTRACTION

AVS Modules Moravec (6)

NAME

Moravec - performs Moravec Interest Operator point feature extraction

SYNOPSIS

Name Moravec

Availability ShapeInference module library

Type filter

Inputs field 2D 2-space 1-vector uniform byte (REQUIRED)
field 2D 2-space 1-vector uniform byte (OPTIONAL)

Outputs field 2D 2-vector uniform float
field 2D 2-vector uniform float
byte

Parameters	Name	Type	Default	Min	Max
	Maxima Only	boolean	1	0	1
	FPs (fraction)	real	0.001	0	100

DESCRIPTION

This module implements a point based feature extraction algorithm based upon the Moravec Interest Operator. This operator defines an interest value for each point being the minimum variance as measured along the four major directions of a five by five local neighbourhood centred on the pixel of interest. For our purposes the variance is defined as the sum of the squares of the difference between each point on the diagonal and the central point of interest. Only a number of points specified by the parameters "Maxima Only" and

"FPs (fraction)" are passed to the output, with the remaining points being annulled. The choice of which points to include is determined by iteratively adjusting some threshold until the required number of interest points remain.

INPUTS

Data Field (required; field 2D 2-space 1-vector uniform byte)

Data Field (optional; field 2D 2-space 1-vector uniform byte)

The inputs are a pair of "single band" grey scale images. It is assumed that the images exhibit similar characteristics, i.e. they have the same image dimensions. Otherwise, these images are processed in isolation.

PARAMETERS

Maxima Only

This is a switch which determines whether points of local maxima of Moravec Interest are included as possible features.

FPs (fraction)

This value determines the approximate number of feature points required at the output as a percentage fraction of the number of pixels in the original image.

OUTPUTS

Data Field (required; field 2D 2-vector uniform float)

Data Field (optional; field 2D 2-vector uniform float)

These fields contains copies of the left and right input images in the first "band" (i.e. vector element 0) respectively, together with Moravec Interest point information in the last "band" (i.e. vector element 1).

Byte

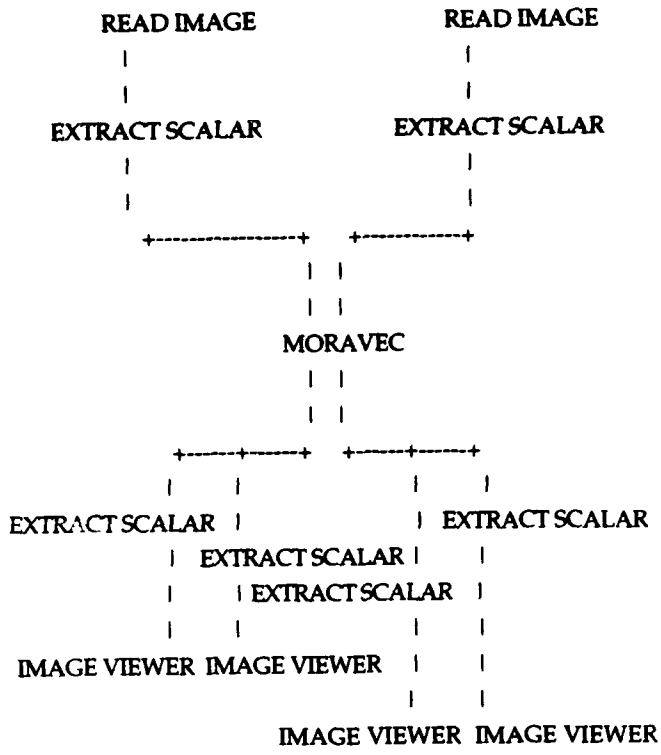
This indicates the type of feature being output.

EXAMPLE

This example shows the left and right images read in by means of two "read image" modules and converted to "single band" grey images (for the purposes of this example it is assumed here that images are in AVS image format (4 band) hence it is necessary to convert each to a scalar field (i.e. single band) - other input file formats may be read with alternate data input modules (e.g. "ip read vff") and may not require this step).

The input fields are passed to the "Moravec" module which in turn outputs two 2 band fields containing copies of the input images (converted to type float) and the Moravec Interest features.

The extract scalar modules extract each of the bands from each of the images and pass them as grey level images into the image viewer. They could just as easily have been passed as the input images to a matching algorithm such as "barnthom" (written by James Kershaw as part of the ShapeInference library).



RELATED MODULES

Barnthom, FEX.

SEE ALSO

Release 5 AVS Moravec(6)

AVS ON-LINE HELP PAGE FOR FEX FEATURE EXTRACTION

AVS Modules Fex (6)

NAME

Fex - performs an edge based feature extraction

SYNOPSIS

Name Fex

Availability ShapeInference module library

Type filter

Inputs field 2D 2-space 1-vector uniform byte (REQUIRED)
field 2D 2-space 1-vector uniform byte (REQUIRED)

Outputs field 2D 5-vector uniform float
byte

Parameters	Name	Type	Default	Min	Max
	Threshold	integer	50	1	255
	Min Length	integer	5	1	150

DESCRIPTION

This module implements a feature extraction algorithm based on one developed by A. Etemadi. This operator chains together connected edge pixels into strings and then segments these strings into straight line segments. These segments are parameterised by calculating a length, angle and variance of edge strength (both across and along the edge segment). The "threshold" is an input threshold which limits the number of pixels considered : only pixels with an edge strength greater than this value are accepted. The "Min Length" is the minimum number of pixels which must be linked in a chain before the chain is considered. All of the points of a straight edge segment are given the same values, and these are written into the second and successive "bands" of the output field. The original image (first input) is copied to the first band of the output. The second input is an edge filtered image (as in a Sobel or Canny edge filtered output).

INPUTS

Data Field (required; field 2D 2-space 1-vector uniform byte)

Data Field (required; field 2D 2-space 1-vector uniform byte)

The inputs are a pair of "single band" grey scale images. The first input is assumed to be the original grey level image. The second input is assumed to be the edge filtered version of the original image (note that this edge filtered version should have thinned edges).

PARAMETERS

Threshold

This value determines the minimum edge strength of the edge filtered image that is required of a pixel before it is considered to be an edge point.

Min Length

This value determines the minimum chain length (in pixels) before that chain is considered a valid edge.

OUTPUTS

Data Field (required; field 2D 5-vector uniform float)

This field contains a copy of the original input image in the *first* "band" (*i.e. vector element 0*), while the remaining bands (i.e. vector elements 1..4) contain the parameters of the edge segment that pixel belongs to.

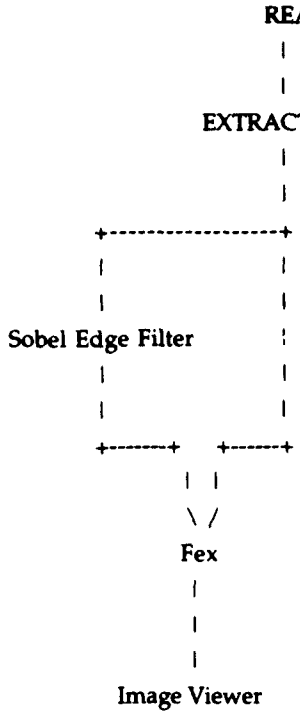
Byte

This value indicates the type of feature which has been extracted.

EXAMPLE

This example shows the original image being read in by the "read image" module and then one of the bands being extracted to give a grey level image (this extraction is necessary due to the four band standard AVS image format). The image is then passed to the first input of the "Fex" module and the "Sobel" edge filtering module. Fex in turn takes the original grey level image and the edge filtered image before passing the output to the "Image Viewer" which is able to take multiple dimension floating point images.

The output could also have been connected to one of the inputs of a feature matching algorithm such as "Barnthom" which is also available in the ShapeInference library.



RELATED MODULES

Barnthom, Moravec.

SEE ALSO

Release	5	AVS	Fex(6)

AVS ON-LINE HELP PAGE FOR INIT_RELAX(FOCUS_CORR)

AVS Modules Init_Relax(focus_corr) (6)

NAME

Init_Relax(focus_corr) - performs relaxation initialisation for various feature sets.

SYNOPSIS

Name Init_Relax(focus_corr)

Availability ShapeInference module library

Type Filter

Inputs field 2D uniform float (REQUIRED)
field 2D uniform float (REQUIRED)
field 2D uniform 4-vector float (OPTIONAL)

Outputs field 2D 2-space 4-vector uniform float
field 2D 3-vector float

Parameters

Name	Type	Default	Min	Max
Centre_x	float	0.000	unbound	unbound
Centre_y	float	0.000	unbound	unbound
Clock_Wise_Rot	float	0.000	0.000	360.0
Trans_x	float	0.000	unbound	unbound
Trans_y	float	0.000	unbound	unbound
Search_Window	int	10	1	unbound

DESCRIPTION

This module implements an initialisation for a relaxation matching scheme. It takes two feature fields (of the same type) and identifies probable matches between the two feature sets, with an initial probability of match. The possible matches are determined as any match within <Search_Window> pixels of the centre of the search area. The centre of the search area is determined in the first instance by the location of the primary feature (in the first input feature field) added to the local estimate of flow for the primary feature (as indicated by the optional flow field input). If there is no optional input, or if the values in the input flow field indicate that no local estimate of flow has been determined, the centre of the search area is

determined by the conformal transform described by the first five parameters of the location of the primary feature.
The associated initial probability of each possible match is determined by finding the discrete co-variance between a five by five local mask centred on the location of each of the two potentially matched feature points.
The first output is a flow field which holds copies of the two initial images (in the first band of the incoming feature images) in its first two bands. The second output is an array which holds lists of features, with associated matches and probabilities. The output is interpretable by the relaxation matching algorithms.

INPUTS

Data Field (required; field 2D uniform float)

Data Field (required; field 2D uniform float)

These inputs are a pair of similar feature fields, where the first "band" holds the original image and the remaining bands hold the feature vectors (one element per band at the feature position).
The feature fields have to have the same depth (or length of vector).

Data Field (optional; field 2D 4-vector uniform float)

This input is an optional field which specifies local flow information where this information is available.

PARAMETERS

Centre_x

This is the centre of the two dimensional rotation (horizontal)

Centre_y

This is the vertical centre of the two dimensional global flow rotation estimate.

Clock_Wise_Rot

This is the magnitude of the global rotation estimate.

Trans_x

This is the global horizontal translation estimate.

Trans_y

This is the global vertical translation estimate.

Search_Window

This is the magnitude of the search area, specified as the number pixels from the centre of the search area.

OUTPUTS

Data Field (required; field 2D 4-vector uniform float)

These fields contains copies of the left and right input images in the first "bands" (i.e. vector elements 0 and 1 respectively). Bands 2 and 3 are empty.

Data Field (required; field 2D float)

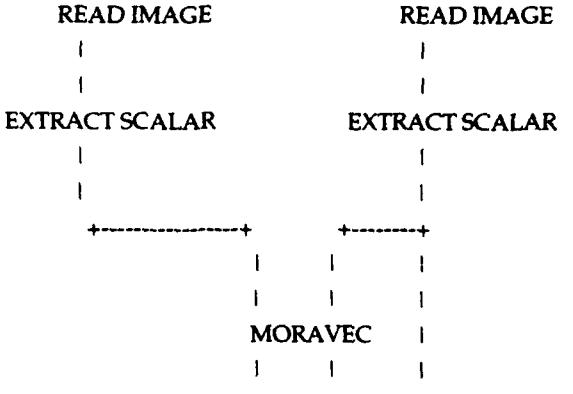
This field holds the lists of matches. The first index indicates the primary feature, the second index indicates the possible match number and the position in the vector element contains information such as x displacement, y displacement and initial probability.

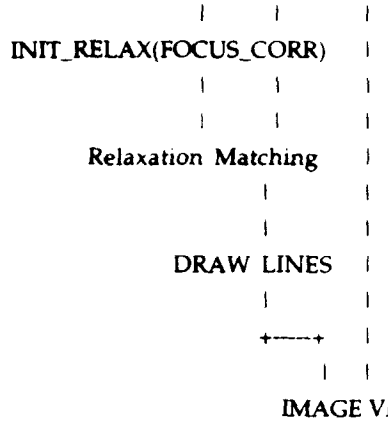
EXAMPLE

This example shows the left and right images read in by means of two "read image" modules and converted to "single band" grey images (for the purposes of this example it is assumed here that images are in AVS image format (4 band) hence it is necessary to convert each to a scalar field (i.e. single band) - other input file formats may be read with alternate data input modules (e.g. "ip read vff") and may not require this step).

The input fields are passed to the "Moravec" module (written by James Kershaw for the Shape Inference library) which in turn outputs two 2 band fields containing copies of the input images (converted to type float) and the Moravec Interest features.

These Moravec feature fields are then passed to the input of the "Init_Relax(focus_corr)" module which carries out the initialisation and calculation of possible matches, before passing a flow field and probability table to a relaxation matching algorithm which in turn passes a flow field to the "draw lines" module (written by Garry Hamlyn for Shape Inference), which enables the Image Viewer to display the flow vectors on the original image (supplied independently).





RELATED MODULES

MORAVEC, FEX, DRAW LINES, HANNAH, RELAXATION

SEE ALSO

Release	5	AVS	Init_Relax(Focus_corr)(6)

AVS ON-LINE HELP PAGE FOR INIT_RELAX(FOCUS_GEN)

AVS Modules Init_Relax(focus_gen) (6)

NAME

Init_Relax(focus_gen) - performs relaxation initialisation for various feature sets.

SYNOPSIS

Name Init_Relax(focus_gen)

Availability ShapeInference module library

Type Filter

Inputs field 2D uniform float (REQUIRED)
field 2D uniform float (REQUIRED)
field 2D uniform 4-vector float (OPTIONAL)

Outputs field 2D 2-space 4-vector uniform float
field 2D 3-vector float

Parameters

Name	Type	Default	Min	Max
Centre_x	float	0.000	unbound	unbound
Centre_y	float	0.000	unbound	unbound
Clock_Wise_Rot	float	0.000	0.000	360.0
Trans_x	float	0.000	unbound	unbound
Trans_y	float	0.000	unbound	unbound
Search_Window	int	10	1	unbound

DESCRIPTION

This module implements an initialisation for a relaxation matching scheme. It takes two feature fields (of the same type) and identifies probable matches between the two feature sets, with an initial probability of match. The possible matches are determined as any match within <Search_Window> pixels of the centre of the search area. The centre of the search area is determined in the first instance by the location of the primary feature (in the first input feature field) added to the local estimate of flow for the primary feature (as indicated by the optional flow field input). If there is no optional input, or if the values in the input flow field indicate that no local estimate of flow has been determined, the centre of the search area is determined by the conformal transform described by the first five parameters

of the location of the primary feature.
The associated initial probability of each possible match is determined by finding the Euclidean distance between each of the two potentially matched feature points.
The first output is a flow field which holds copies of the two initial images (in the first band of the incoming feature images) in its first two bands. The second output is an array which holds lists of features, with associated matches and probabilities. The output is interpretable by the relaxation matching algorithms.

INPUTS

Data Field (required; field 2D uniform float)
Data Field (required; field 2D uniform float)

These inputs are a pair of similar feature fields, where the first "band" holds the original image and the remaining bands hold the feature vectors (one element per band at the feature position).
The feature fields have to have the same depth (or length of vector).

Data Field (optional; field 2D 4-vector uniform float)

This input is an optional field which specifies local flow information where this information is available.

PARAMETERS

Centre_x

This is the centre of the two dimensional rotation (horizontal)

Centre_y

This is the vertical centre of the two dimensional global flow rotation estimate.

Clock_Wise_Rot

This is the magnitude of the global rotation estimate.

Trans_x

This is the global horizontal translation estimate.

Trans_y

This is the global vertical translation estimate.

Search_Window

This is the magnitude of the search area, specified as the number pixels from the centre of the search area.

OUTPUTS

Data Field (required; field 2D 4-vector uniform float)

These fields contains copies of the left and right input images in the first "bands" (i.e. vector elements 0 and 1 respectively). Bands 2 and 3 are empty.

Data Field (required; field 2D float)

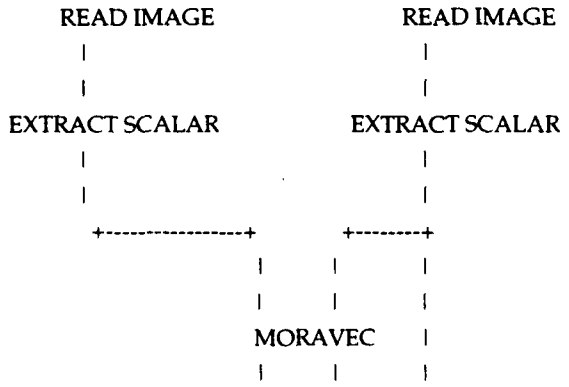
This field holds the lists of matches. The first index indicates the primary feature, the second index indicates the possible match number and the position in the vector element contains information such as x displacement, y displacement and initial probability.

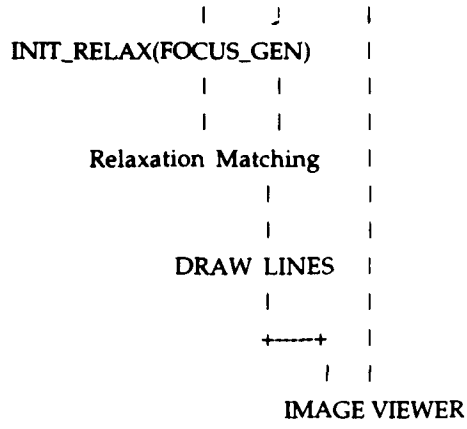
EXAMPLE

This example shows the left and right images read in by means of two "read image" modules and converted to "single band" grey images (for the purposes of this example it is assumed here that images are in AVS image format (4 band) hence it is necessary to convert each to a scalar field (i.e. single band) - other input file formats may be read with alternate data input modules (e.g. "ip read vff") and may not require this step).

The input fields are passed to the "Moravec" module (written by James Kershaw for the Shape Inference library) which in turn outputs two 2 band fields containing copies of the input images (converted to type float) and the Moravec Interest features.

These Moravec feature fields are then passed to the input of the "Init_Relax(focus_gen)" module which carries out the initialisation and calculation of possible matches, before passing a flow field and probability table to a relaxation matching algorithm which in turn passes a flow field to the "draw lines" module (written by Garry Hamlyn for Shape Inference), which enables the Image Viewer to display the flow vectors on the original image (supplied independently).





RELATED MODULES

MORAVEC, FEX, DRAW LINES, HANNAH, RELAXATION

SEE ALSO

Release	5	AVS	Init_Relax(Focus_Gen)(6)

AVS ON-LINE HELP PAGE FOR INIT_RELAX(NN_CORR)

AVS Modules Init_Relax(NN_corr) (6)

NAME

Init_Relax(NN_corr) - performs relaxation initialisation for various feature sets.

Name Init_Relax(NN_corr)

Availability ShapeInference module library

Type Filter

Inputs field 2D uniform float (REQUIRED)
field 2D uniform float (REQUIRED)
field 2D uniform 4-vector float (OPTIONAL)

Outputs field 2D 2-space 4-vector uniform float
field 2D 3-vector float

Parameters

Name	Type	Default	Min	Max
Centre_x	float	0.000	unbound	unbound
Centre_y	float	0.000	unbound	unbound
Clock_Wise_Rot	float	0.000	0.000	360.0
Trans_x	float	0.000	unbound	unbound
Trans_y	float	0.000	unbound	unbound
Search_Window	int	10	1	unbound

DESCRIPTION

This module implements an initialisation for a relaxation matching scheme. It takes two feature fields (of the same type) and identifies probable matches between the two feature sets, with an initial probability of match. The possible matches are determined as the closest <Search_Window> features to the search area. The centre of the search area is determined in the first instance by the location of the primary feature (in the first input feature field) added to the local estimate of flow for the primary feature (as indicated by the optional flow field input). If there is no optional input, or if the values in the input flow field indicate that no local estimate of flow has been determined, the centre of the search area is determined by the conformal transform described by the first five parameters

of the location of the primary feature.
The associated initial probability of each possible match is determined by finding the discrete co-variance between a five by five local mask centred on the location of each of the two potentially matched feature points.
The first output is a flow field which holds copies of the two initial images (in the first band of the incoming feature images) in its first two bands. The second output is an array which holds lists of features, with associated matches and probabilities. The output is interpretable by the relaxation matching algorithms.

INPUTS

Data Field (required; field 2D uniform float)

Data Field (required; field 2D uniform float)

These inputs are a pair of similar feature fields, where the first "band" holds the original image and the remaining bands hold the feature vectors (one element per band at the feature position). The feature fields have to have the same depth (or length of vector).

Data Field (optional; field 2D 4-vector uniform float)

This input is an optional field which specifies local flow information where this information is available.

PARAMETERS

Centre_x

This is the centre of the two dimensional rotation (horizontal)

Centre_y

This is the vertical centre of the two dimensional global flow rotation estimate.

Clock_Wise_Rot

This is the magnitude of the global rotation estimate.

Trans_x

This is the global horizontal translation estimate.

Trans_y

This is the global vertical translation estimate.

Search_Window

This is the magnitude of the search area, specified as the number of matches to consider (in order of closeness)

OUTPUTS

Data Field (required; field 2D 4-vector uniform float)

These fields contains copies of the left and right input images in the first "bands" (i.e. vector elements 0 and 1 respectively). Bands 2 and 3 are empty.

Data Field (required, field 2D float)

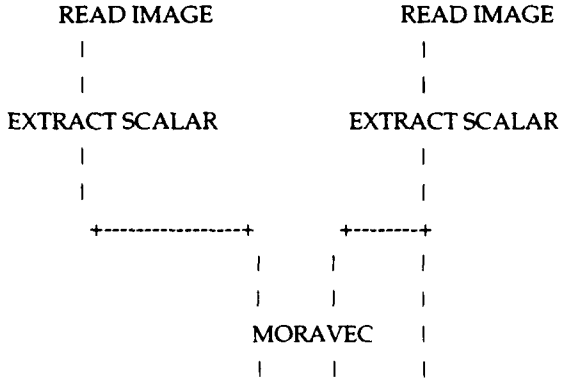
This field holds the lists of matches. The first index indicates the primary feature, the second index indicates the possible match number and the position in the vector element contains information such as x displacement, y displacement and initial probability.

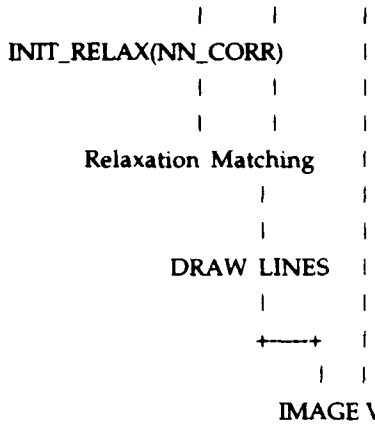
EXAMPLE

This example shows the left and right images read in by means of two "read image" modules and converted to "single band" grey images (for the purposes of this example it is assumed here that images are in AVS image format (4 band) hence it is necessary to convert each to a scalar field (i.e. single band) - other input file formats may be read with alternate data input modules (e.g. "ip read vff") and may not require this step).

The input fields are passed to the "Moravec" module (written by James Kershaw for the Shape Inference library) which in turn outputs two 2 band fields containing copies of the input images (converted to type float) and the Moravec Interest features.

These Moravec feature fields are then passed to the input of the "Init_Relax(NN_corr)" module which carries out the initialisation and calculation of possible matches, before passing a flow field and probability table to a relaxation matching algorithm which in turn passes a flow field to the "draw lines" module (written by Garry Hamlyn for Shape Inference), which enables the Image Viewer to display the flow vectors on the original image (supplied independently).





RELATED MODULES

MORAVEC, FEX, DRAW LINES, HANNAH, RELAXATION

SEE ALSO

Release	5	AVS	Init_Relax(NN_corr)(6)

AVS ON-LINE HELP PAGE FOR INIT_RELAX(NN_GEN)

AVS Modules Init_Relax(NN_gen) (6)

NAME

Init_Relax(NN_gen) - performs relaxation initialisation for various feature sets.

SYNOPSIS

Name Init_Relax(NN_gen)

Availability ShapeInference module library

Type Filter

Inputs field 2D uniform float (REQUIRED)
field 2D uniform float (REQUIRED)
field 2D uniform 4-vector float (OPTIONAL)

Outputs field 2D 2-space 4-vector uniform float
field 2D 3-vector float

Parameters

Name	Ty	Default	Min	Max
Centre_x	float	0.000	unbound	unbound
Centre_y	float	0.000	unbound	unbound
Clock_Wise_Rot	float	0.000	0.000	360.0
Trans_x	float	0.000	unbound	unbound
Trans_y	float	0.000	unbound	unbound
Search_Window	int	10	1	unbound

DESCRIPTION

This module implements an initialisation for a relaxation matching scheme. It takes two feature fields (of the same type) and identifies probable matches between the two feature sets, with an initial probability of match. The possible matches are determined as the closest <Search_Window> features to the search area. The centre of the search area is determined in the first instance by the location of the primary feature (in the first input feature field) added to the local estimate of flow for the primary feature (as indicated by the optional flow field input). If there is no optional input, or if the values in the input flow field indicate that no local estimate of flow has been determined, the centre of the search area is determined by the conformal transform described by the first five parameters

of the location of the primary feature.
The associated initial probability of each possible match is determined by finding the Euclidean distance between each of the two potentially matched feature points.
The first output is a flow field which holds copies of the two initial images (in the first band of the incoming feature images) in its first two bands. The second output is an array which holds lists of features, with associated matches and probabilities. The output is interpretable by the relaxation matching algorithms.

INPUTS

Data Field (required; field 2D uniform float)

Data Field (required; field 2D uniform float)

These inputs are a pair of similar feature fields, where the first "band" holds the original image and the remaining bands hold the feature vectors (one element per band at the feature position).
The feature fields have to have the same depth (or length of vector).

Data Field (optional; field 2D 4-vector uniform float)

This input is an optional field which specifies local flow information where this information is available.

PARAMETERS

Centre_x

This is the centre of the two dimensional rotation (horizontal)

Centre_y

This is the vertical centre of the two dimensional global flow rotation estimate.

Clock_Wise_Rot

This is the magnitude of the global rotation estimate.

Trans_x

This is the global horizontal translation estimate.

Trans_y

This is the global vertical translation estimate.

Search_Window

This is the magnitude of the search area, specified as the number of matches to consider (in order of closeness)

OUTPUTS

Data Field (required; field 2D 4-vector uniform float)

These fields contains copies of the left and right input images in the first "bands" (i.e. vector elements 0 and 1 respectively). Bands 2 and 3 are empty.

Data Field (required; field 2D float)

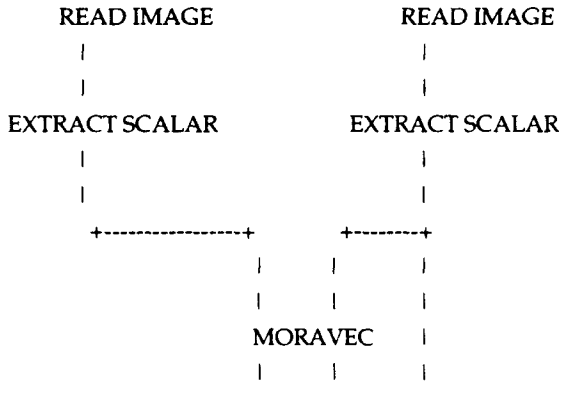
This field holds the lists of matches. The first index indicates the primary feature, the second index indicates the possible match number and the position in the vector element contains information such as x displacement, y displacement and initial probability.

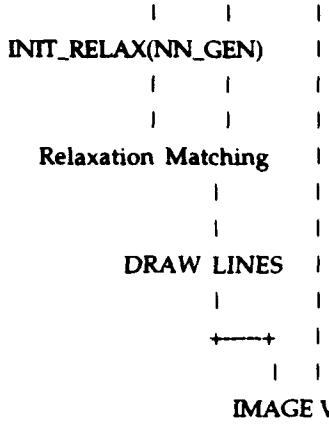
EXAMPLE

This example shows the left and right images read in by means of two "read image" modules and converted to "single band" grey images (for the purposes of this example it is assumed here that images are in AVS image format (4 band) hence it is necessary to convert each to a scalar field (i.e. single band) - other input file formats may be read with alternate data input modules (e.g. "ip read vff") and may not require this step).

The input fields are passed to the "Moravec" module (written by James Kershaw for the Shape Inference library) which in turn outputs two 2 band fields containing copies of the input images (converted to type float) and the Moravec Interest features.

These Moravec feature fields are then passed to the input of the "Init_Relax(NN_gen)" module which carries out the initialisation and calculation of possible matches, before passing a flow field and probability table to a relaxation matching algorithm which in turn passes a flow field to the "draw lines" module (written by Garry Hamlyn for Shape Inference), which enables the Image Viewer to display the flow vectors on the original image (supplied independently).





RELATED MODULES

MORAVEC, FEX, DRAW LINES, HANNAH, RELAXATION

SEE ALSO

Release	5	AVS	Init_Relax(NN_gen)(6)

AVS ON-LINE HELP PAGE FOR RELAXATION

AVS Modules Relaxation (6)

NAME

Relaxation - performs relaxation matching for a given initialised table.

SYNOPSIS

Name Relaxation

Availability ShapeInference module library

Type filter

Inputs field 2D 4-vector uniform float (REQUIRED)
field 2D 3-vector float (REQUIRED)
field 2D uniform float (OPTIONAL)

Outputs field 2D 2-space 4-vector uniform float

Parameters	Name	Type	Default	Min	Max
	Neighbourhood	integer	6	1	100
	Iterations	integer	5	0	20

DESCRIPTION

This module implements a relaxation matching scheme capable of taking an initialised probability table (field 2D 3-vector) which has primary features along the first dimension, potential matches along the second dimension and values of displacement and initial estimate of match probability contained in the vector element, (** The array is more specific than that and is described in the source code comments **), and determining a sparse flow field, with an estimate of flow calculated for each of the primary features.

The result is a single flow field which contains copies of the left and right original images (in vector positions 0,1) and a set of displacements (vertical and horizontal) between the matched feature pairs (vector positions 2,3 respectively).

The relaxation scheme is based on neighbourhood support, where the neighbourhood is calculated as any primary feature within <Neighbourhood> pixels of the primary feature (not including the primary feature in question). The iterative relaxation algorithm is performed <Iterations> times.

INPUTS

Data Field (required; field 2D 4-vector uniform float)

This input holds the two original images in its first two bands. The remaining two bands are empty.

Data Field (required; field 2D 3-vector float)

This input holds the initial probability estimates with other relevant information, as follows.

Field[i][0] holds {Xlocation,Ylocation,Number of Matches} of the primary feature 'i'

Field[i][1] holds {0,0,Prob(NoMatch)} for feature 'i'

Field[i][j] holds {dX,dY,Prob} for the 'jth' match of feature 'i', where {dX,dY} are displacements from feature 'i', and {Prob} is the probability (estimate) that the 'jth' match feature should be matched with the 'ith' feature.

Data Field (optional; field 2D uniform float)

This input is an optional feature field which may be used to modify the neighbourhood search area. It is meaningless at the moment (i.e. it is ignored).

PARAMETERS

Neighbourhood

This value determines the size of the neighbourhood (in pixels from the primary feature).

Iterations

This determines the number of times the relaxation calculation is performed.

OUTPUTS

Data Field (required; field 2D 4-vector uniform float)

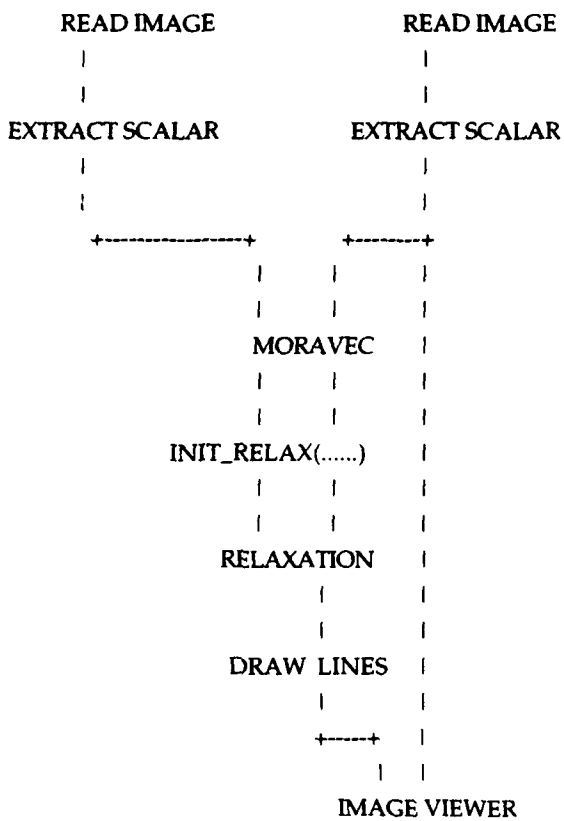
These fields contains copies of the left and right input images in the first "bands" (i.e. vector elements 0 and 1 respectively) together with vertical and horizontal disparity information in the last two "bands" (i.e. vector elements 2 and 3 respectively).

EXAMPLE

This example shows the left and right images read in by means of two "read image" modules and converted to "single band" grey images (for the purposes of this example it is assumed here that images are in AVS image format (4 band) hence it is necessary to convert each to a scalar field (i.e. single band) - other input file formats may be read with alternate data input modules (e.g. "ip read vff") and may not require this step).

The input fields are passed to the "Moravec" module (written by James Kershaw for the Shape Inference library) which in turn outputs two 2 band fields containing copies of the input images (converted to type float) and the Moravec Interest features.

These Moravec feature fields are then passed to an "Init_Relax...." module which sets up the probability table for relaxation. This table is then input to "Relaxation" which carries out the flow calculation before passing a flow field to the "draw lines" module (written by Garry Hamlyn for Shape Inference), which enables the Image Viewer to display the flow vectors on the original image (supplied independently).



RELATED MODULES

MORAVEC, FEX, DRAW LINES, HANNAH, INIT_RELAX(...)

SEE ALSO

Release	5	AVS	Relaxation (6)

AVS ON-LINE HELP PAGE FOR RELAXATION_NN

AVS Modules Relaxation_NN (6)

NAME

Relaxation_NN - performs relaxation matching for a given initialised table.

SYNOPSIS

Name Relaxation_NN

Availability ShapeInference module library

Type filter

Inputs field 2D 4-vector uniform float (REQUIRED)
field 2D 3-vector float (REQUIRED)
field 2D uniform float (OPTIONAL)

Outputs field 2D 2-space 4-vector uniform float

Parameters	Name	Type	Default	Min	Max
	Neighbourhood	integer	6	1	100
	Iterations	integer	5	0	20

DESCRIPTION

This module implements a relaxation matching scheme capable of taking an initialised probability table (field 2D 3-vector) which has primary features along the first dimension, potential matches along the second dimension and values of displacement and initial estimate of match probability contained in the vector element, (** The array is more specific than that and is described in the source code comments **), and determining a sparse flow field, with an estimate of flow calculated for each of the primary features.

The result is a single flow field which contains copies of the left and right original images (in vector positions 0,1) and a set of displacements (vertical and horizontal) between the matched feature pairs (vector positions 2,3 respectively).

The relaxation scheme is based on neighbourhood support, where the neighbourhood is calculated to be the <Neighbourhood> closest neighbours to the primary feature (not including the primary feature in question).

The iterative relaxation algorithm is performed <Iterations> times.

INPUTS

Data Field (required; field 2D 4-vector uniform float)

This input holds the two original images in its first two bands. The remaining two bands are empty.

Data Field (required; field 2D 3-vector float)

This input holds the initial probability estimates with other relevant information. as follows.

Field[i][0] holds {Xlocation,Ylocation,Number of Matches} of the primary feature 'i'

Field[i][1] holds {0,0,Prob(NoMatch)} for feature 'i'

Field[i][j] holds {dX,dY,Prob} for the 'jth' match of feature 'i', where {dX,dY} are displacements from feature 'i', and {Prob} is the probability (estimate) that the 'jth' match feature should be matched with the 'ith' feature.

Data Field (optional; field 2D uniform float)

This input is an optional feature field which may be used to modify the neighbourhood search area. It is meaningless at the moment (i.e. it is ignored).

PARAMETERS

Neighbourhood

This value determines the size of the neighbourhood in terms of the number of closest neighbours.

Iterations

This determines the number of times the relaxation calculation is performed.

OUTPUTS

Data Field (required; field 2D 4-vector uniform float)

These fields contains copies of the left and right input images in the first "bands" (i.e. vector elements 0 and 1 respectively) together with vertical and horizontal disparity information in the last two "bands" (i.e. vector elements 2 and 3 respectively).

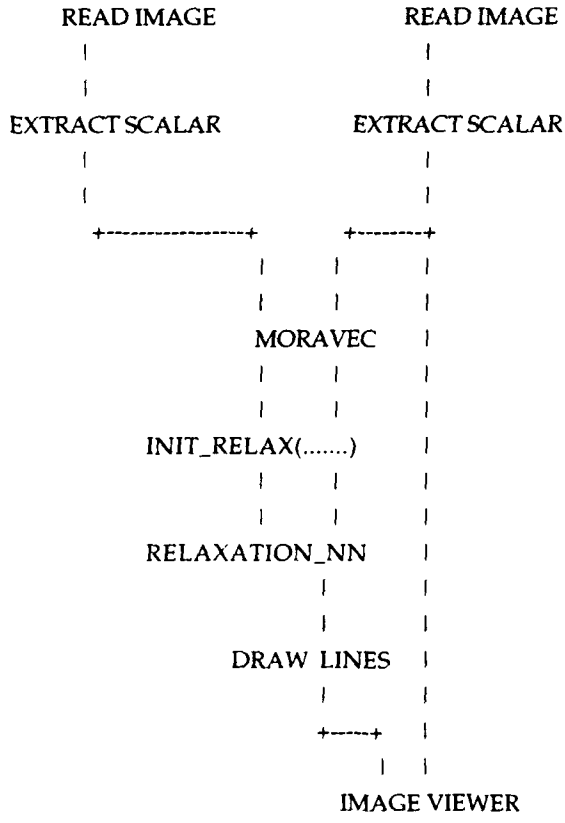
EXAMPLE

This example shows the left and right images read in by means of two "read image" modules and converted to "single band" grey images (for the

purposes of this example it is assumed here that images are in AVS image format (4 band) hence it is necessary to convert each to a scalar field (i.e. single band) - other input file formats may be read with alternate data input modules (e.g. "ip read vff") and may not require this step).

The input fields are passed to the "Moravec" module (written by James Kershaw for the Shape Inference library) which in turn outputs two 2 band fields containing copies of the input images (converted to type float) and the Moravec Interest features.

These Moravec feature fields are then passed to an "Init_Relax...." module which sets up the probability table for relaxation. This table is then input to "Relaxation_NN" which carries out the flow calculation before passing a flow field to the "draw lines" module (written by Garry Hamlyn for Shape Inference), which enables the Image Viewer to display the flow vectors on the original image (supplied independently).



RELATED MODULES

MORAVEC, FEX, DRAW LINES, HANNAH, INIT_RELAX(...)

SEE ALSO

Release	5	AVS	Relaxation_NN (6)

AVS ON-LINE HELP PAGE FOR WGTD_RELAX

AVS Modules Wgtd_Relax (6)

NAME

Wgtd_Relax - performs relaxation matching for a given initialised table.

SYNOPSIS

Name Wgtd_Relax

Availability ShapeInference module library

Type filter

Inputs field 2D 4-vector uniform float (REQUIRED)
field 2D 3-vector float (REQUIRED)
field 2D uniform float (OPTIONAL)

Outputs field 2D 2-space 4-vector uniform float

Parameters	Name	Type	Default	Min	Max
	Neighbourhood	integer	6	1	100
	Iterations	integer	5	0	20

DESCRIPTION

This module implements a relaxation matching scheme capable of taking an initialised probability table (field 2D 3-vector) which has primary features along the first dimension, potential matches along the second dimension and values of displacement and initial estimate of match probability contained in the vector element, (** The array is more specific than that and is described in the source code comments **), and determining a sparse flow field, with an estimate of flow calculated for each of the primary features.

The result is a single flow field which contains copies of the left and right original images (in vector positions 0,1) and a set of displacements (vertical and horizontal) between the matched feature pairs (vector positions 2,3 respectively).

The relaxation scheme is based on neighbourhood support, where the neighbourhood is calculated as any primary feature within <Neighbourhood> pixels of the primary feature (not including the primary feature in question). The support provided is dependent upon the distance between the two features (feature and

neighbour) and is Gaussian in shape with a variance such that the largest possible distance between features has a weighting of "0.1" and the closest possible distance between features (0) has a weighting of "1.0".
The iterative relaxation algorithm is performed <Iterations> times.

INPUTS

Data Field (required; field 2D 4-vector uniform float)

This input holds the two original images in its first two bands. The remaining two bands are empty.

Data Field (required; field 2D 3-vector float)

This input holds the initial probability estimates with other relevant information, as follows.
Field[i][0] holds {Xlocation,Ylocation,Number of Matches} of the primary feature 'i'
Field[i][1] holds {0,0,Prob(NoMatch)} for feature 'i'
Field[i][j] holds {dX,dY,Prob} for the 'jth' match of feature 'i', where {dX,dY} are displacements from feature 'i', and {Prob} is the probability (estimate) that the 'jth' match feature should be matched with the 'ith' feature.

Data Field (optional; field 2D uniform float)

This input is an optional feature field which may be used to modify the neighbourhood search area. It is meaningless at the moment (i.e. it is ignored).

PARAMETERS

Neighbourhood

This value determines the size of the neighbourhood (in pixels from the primary feature).

Iterations

This determines the number of times the relaxation calculation is performed.

OUTPUTS

Data Field (required; field 2D 4-vector uniform float)

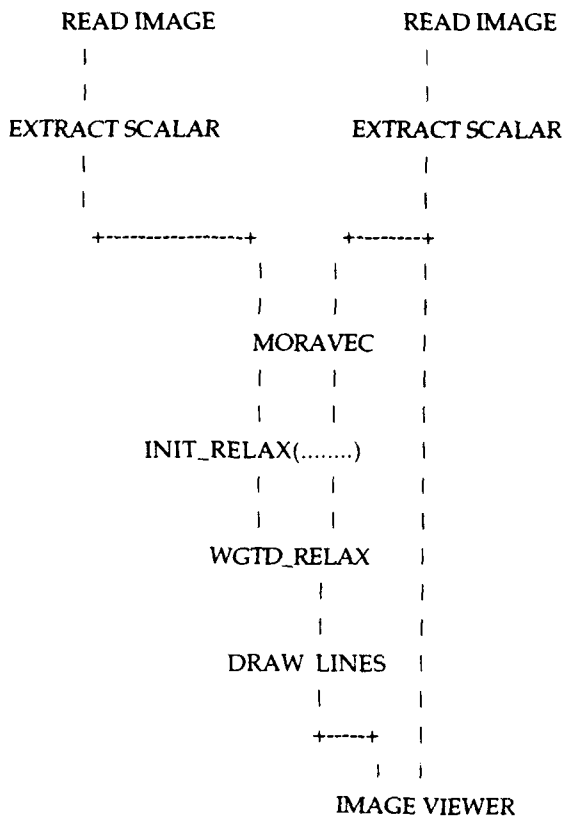
These fields contains copies of the left and right input images in the first "bands" (i.e. vector elements 0 and 1 respectively) together with vertical and horizontal disparity information in the last two "bands" (i.e. vector elements 2 and 3 respectively).

EXAMPLE

This example shows the left and right images read in by means of two "read image" modules and converted to "single band" grey images (for the purposes of this example it is assumed here that images are in AVS image format (4 band) hence it is necessary to convert each to a scalar field (i.e. single band) - other input file formats may be read with alternate data input modules (e.g. "ip read vff") and may not require this step).

The input fields are passed to the "Moravec" module (written by James Kershaw for the Shape Inference library) which in turn outputs two 2 band fields containing copies of the input images (converted to type float) and the Moravec Interest features.

These Moravec feature fields are then passed to an "Init_Relax...." module which sets up the probability table for relaxation. This table is then input to "Wgtd_Relax" which carries out the flow calculation before passing a flow field to the "draw lines" module (written by Garry Hamlyn for Shape Inference), which enables the Image Viewer to display the flow vectors on the original image (supplied independently).



RELATED MODULES

MORAVEC, FEX, DRAW LINES, HANNAH, INIT_RELAX(...)

SEE ALSO

Release	5	AVS	Wgtd_Relax (6)

AVS ON-LINE HELP PAGE FOR WGTD_RELAX_NN

AVS Modules Wgtd_Relax_NN (6)

NAME

Wgtd_Relax_NN - performs relaxation matching for a given initialised table.

SYNOPSIS

Name Wgtd_Relax_NN

Availability ShapeInference module library

Type filter

Inputs field 2D 4-vector uniform float (REQUIRED)
field 2D 3-vector float (REQUIRED)
field 2D uniform float (OPTIONAL)

Outputs field 2D 2-space 4-vector uniform float

Parameters	Name	Type	Default	Min	Max
	Neighbourhood	integer	6	1	100
	Iterations	integer	5	0	20

DESCRIPTION

This module implements a relaxation matching scheme capable of taking an initialised probability table (field 2D 3-vector) which has primary features along the first dimension, potential matches along the second dimension and values of displacement and initial estimate of match probability contained in the vector element, (** The array is more specific than that and is described in the source code comments **), and determining a sparse flow field, with an estimate of flow calculated for each of the primary features.

The result is a single flow field which contains copies of the left and right original images (in vector positions 0,1) and a set of displacements (vertical and horizontal) between the matched feature pairs (vector positions 2,3 respectively).

The relaxation scheme is based on neighbourhood support, where the neighbourhood is calculated as the <Neighbourhood> closest neighbouring features of the primary feature (not including the primary feature in question). The support provided is dependent upon the order of closeness between the two features

(feature and neighbour) and is Gaussian in shape with a variance such that the last position in the match list has a weighting of "0.1".
The iterative relaxation algorithm is performed <Iterations> times.

INPUTS

Data Field (required; field 2D 4-vector uniform float)

This input holds the two original images in its first two bands. The remaining two bands are empty.

Data Field (required; field 2D 3-vector float)

This input holds the initial probability estimates with other relevant information, as follows.
Field[i][0] holds {Xlocation,Ylocation,Number of Matches} of the primary feature 'i'
Field[i][1] holds {0,0,Prob(NoMatch)} for feature 'i'
Field[i][j] holds {dX,dY,Prob} for the 'jth' match of feature 'i', where {dX,dY} are displacements from feature 'i', and {Prob} is the probability (estimate) that the 'jth' match feature should be matched with the 'ith' feature.

Data Field (optional; field 2D uniform float)

This input is an optional feature field which may be used to modify the neighbourhood search area. It is meaningless at the moment (i.e. it is ignored).

PARAMETERS

Neighbourhood

This value determines the size of the neighbourhood (in number of valid neighbours).

Iterations

This determines the number of times the relaxation calculation is performed.

OUTPUTS

Data Field (required; field 2D 4-vector uniform float)

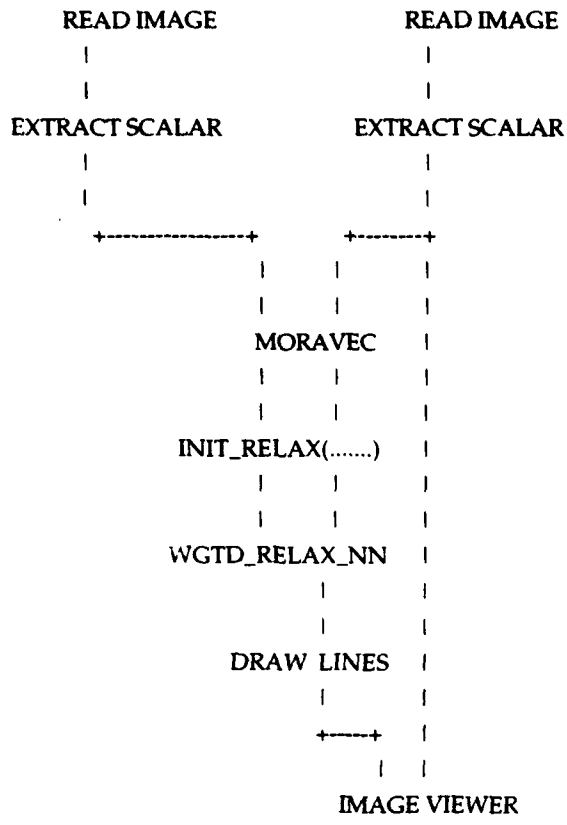
These fields contains copies of the left and right input images in the first "bands" (i.e. vector elements 0 and 1 respectively) together with vertical and horizontal disparity information in the last two "bands" (i.e. vector elements 2 and 3 respectively).

EXAMPLE

This example shows the left and right images read in by means of two "read image" modules and converted to "single band" grey images (for the purposes of this example it is assumed here that images are in AVS image format (4 band) hence it is necessary to convert each to a scalar field (i.e. single band) - other input file formats may be read with alternate data input modules (e.g. "ip read vff") and may not require this step).

The input fields are passed to the "Moravec" module (written by James Kershaw for the Shape Inference library) which in turn outputs two 2 band fields containing copies of the input images (converted to type float) and the Moravec Interest features.

These Moravec feature fields are then passed to an "Init_Relax...." module which sets up the probability table for relaxation. This table is then input to "Wgtd_Relax_NN" which carries out the flow calculation before passing a flow field to the "draw lines" module (written by Garry Hamlyn for Shape Inference), which enables the Image Viewer to display the flow vectors on the original image (supplied independently).



RELATED MODULES

MORAVEC, FEX, DRAW LINES, HANNAH, INIT_RELAX(...)

SEE ALSO

Release	5	AVS	Wgtd_Relax_NN (6)

AVS ON-LINE HELP PAGE FOR HANNAH MATCHING

AVS Modules Hannah matching(6)

NAME

Hannah matching - perform stereo image matching using Hannah algorithm

SYNOPSIS

Name Hannah matching

Availability Shared inference module library

Type filter

Inputs field 2D 2-space 1-vector uniform byte (REQUIRED)
field 2D 2-space 1-vector uniform byte (REQUIRED)

Outputs field 2D 4-vector uniform float

Parameters

Name	Type	Default	Min	Max
window size	integer	11	1	20
num x subimages	integer	6	1	20
num y subimages	integer	6	1	20
num uhm points	integer	1	1	10
int points per cell	integer	1	1	10

DESCRIPTION

This module implements the first stage of a stereo image matching algorithm created by Marsha Jo Hannah, Artificial Intelligence Center, SRI International, and described in PHOTOGRAMMETRIC ENGINEERING AND REMOTE SENSING, Vol 55, No. 12, December 1989, pp 1765-1770. The algorithm essentially uses area based correlation coupled with a variety of hierarchical search techniques including back matching. Only the first stage, which Hannah refers to as "unconstrained hierarchical matching", has been implemented to date. Further stages (not yet implemented) will make use of camera models (known or estimated) and epipolar constraints, culminating with an "anchored matching" technique based on the reliable set of matches obtained by all of the preceding techniques.

INPUTS

Data Field (required; field 2D 2-space 1-vector uniform byte)

Data Field (required; field 2D 2-space 1-vector uniform byte)

The inputs are a pair of "single band" grey scale images. It is assumed that the images exhibit similar characteristics, i.e. they cover approximately the

same area at about the same scale with no major rotation between the images nor any significant time-lapse changes.

PARAMETERS

window size

The size (in pixels) of the correlation window used for the correlation matching performed during "unconstrained hierarchical matching" (first stage of the Hannah algorithm). The default value is set to 11 as suggested by Hannah.

num x subimages

num y subimages

The original input images are divided up into a regular grid of "subimages" or "cells" for subsequent processing. These parameters determine the grid spacings by setting the number of divisions in the *x* and *y* directions. For example, setting "num x subimages" = 6 and "num y subimages" = 4 gives a total of 6 x 4 = 24 subimages. The default values for each are set to 6 as suggested by Hannah.

int points per cell

By passing a statistical "interest operator" over the original images an "interest value" is obtained for each image pixel. The "int points per cell" parameter allows all but the top <int points per cell> of these for each subimage or cell to be disregarded, thus permitting only the most interesting points to be matched. Note that each of these points is a distinct local maxima. The default value has been set to 1 only so that in the first instance the module executes more quickly.

num uhm points

This parameter determines how many of the above mentioned <int points per cell> (in order of merit) will undergo "unconstrained hierarchical matching". The default value has been set to 1 to again enable faster execution in the first instance.

OUTPUTS

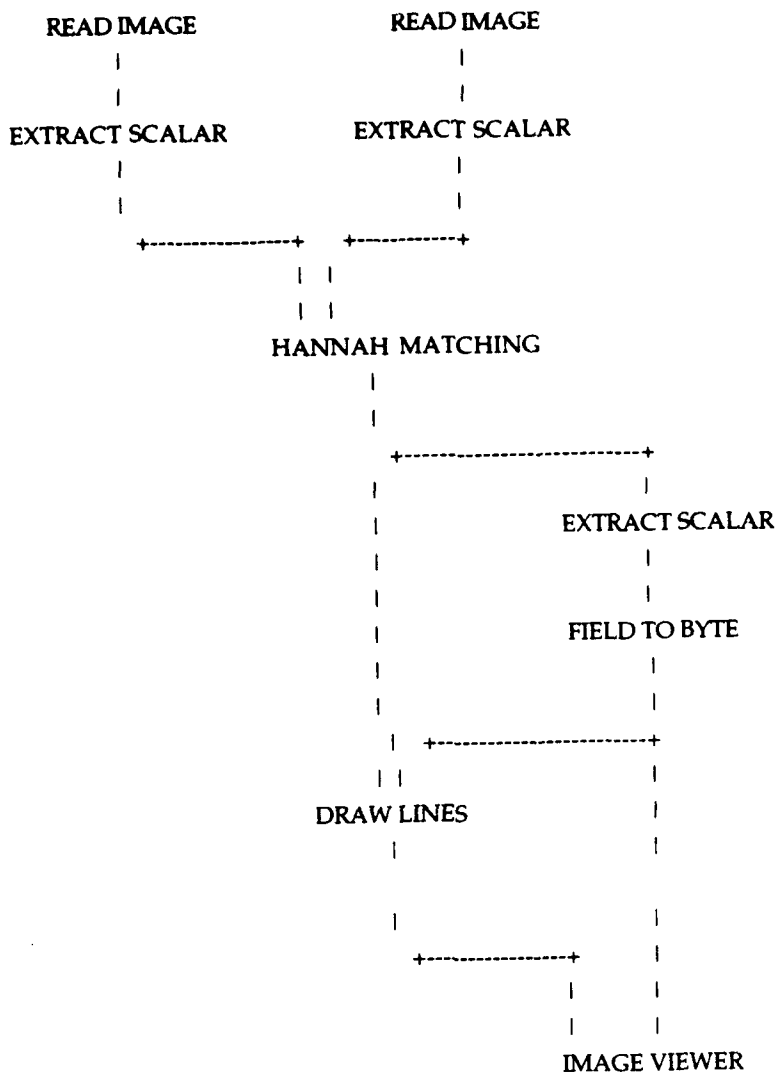
Data Field (required: field 2D 2-space 4-vector uniform float)

This field contains copies of the left and right input images in the first two "bands" (i.e. vector elements 0 and 1), together with pixel *y* and *x* disparity information in the last two "bands" (i.e. vector elements 2 and 3 respectively).

EXAMPLE

This example shows the left and right images read in by means of two "read image" modules and converted to "single band" grey images (for the purposes of this example it is assumed here that images are in AVS image format (4 band) hence it is necessary to convert each to a scalar field (i.e. single band) - other input file formats may be read with alternate data input modules (e.g. "ip read vff") and may not require this step).

The input fields are passed to the "Hannah matching" module which in turn outputs a 4 band field containing copies of the input images (converted to type float) and the x and y direction disparity information. The "draw lines" module extracts the disparity information and uses it to graphically overlay "disparity vectors" onto a copy of one of the original images ("draw lines" was built by Garry Hamlyn DSTO from a variation of a supplied AVS example source code file (/usr/avs/examples/doodle.c) and is also in the "ShapeInference" module library). In this example the copy of the original image is obtained via the Hannah output; it could just as easily have been channelled from one of the data input modules instead. The copy of one of the original images is also passed to the image viewer module for display (note that an upstream feedback connection to the "draw lines" module is required).



RELATED MODULES

draw lines

SEE ALSO

Release 5 AVS Hannah matching(6)

THIS IS A BLANK PAGE

DISTRIBUTION

	Copy No.
Defence Science and Technology Organisation	
Chief Defence Scientist)	
Central Office Executive)	1 shared copy
Counsellor, Defence Science, London	Cont Sht
Counsellor, Defence Science, Washington	Cont Sht
Senior Defence Scientific Adviser	1 copy
Scientific Adviser POLCOM	1 copy
Director, Aeronautical & Maritime Research Laboratory	1 copy
Navy Office	
Naval Scientific Adviser	1 copy
Army Office	
Scientific Adviser, Army	1 copy
Airforce Office	
Air Force Scientific Adviser	1 copy
Defence Intelligence Organisation	
Assistant Secretary Scientific Analysis	1 copy
CSSIP	
Director	1 copy
Assoc. Prof. M. Brooks	1 copy
Mr. M. Baker	1 copy
Mr. Chunping Ding	1 copy
Mr. D. Gibbins	1 copy
Dr. D. Huynh	1 copy
Electronics & Surveillance Research Laboratory	
Director	1 copy
Chief Information Technology Division	1 copy
Chief Electronic Warfare Division	Cont Sht
Chief Guided Weapons Division	"
Chief Communications Division	"
Chief Land, Space and Optoelectronics Division	"
Chief High Frequency Radar Division	"
Chief Microwave Radar Division	"
Chief Air Operations Division	"
Chief Maritime Operations Division	"
Research Leader Command & Control and Intelligence Systems	1 copy
Research Leader Military Computing Systems	1 copy
Research Leader Command, Control and Communications	1 copy
Head, Information Acquisition & Processing Group	1 copy

DISTRIBUTION

	Copy No.
Dr M. Nelson, Information Acquisition & Processing Group	1 copy
Dr. V. Shettigara, Information Acquisition & Processing Group	1 copy
Mr. P. Deer, Information Acquisition & Processing Group	1 copy
Head, Information Management Group	1 copy
Dr. G. Newsam, Information Management Group	1 copy
Dr. N. Redding, Information Management Group	1 copy
Mr. R. Whatmough, Information Management Group	1 copy
Mr. K. Tang, Information Management Group	1 copy
Dr. P. Whitbread, Information Management Group	1 copy
Head Intelligence Systems Group	2 copies
Dr. J. Hildebrandt, Intelligence Systems Group	2 copies
Head Command Support Systems Group	1 copy
James Kershaw (Author), Intelligence Systems Group	2 copies
Garry Hamlyn (Author), Information Management Group	2 copies
Publications & Publicity Officer ITD	1 copy
Libraries and Information Services	
Australian Government Publishing Service	1 copy
Defence Central Library, Technical Reports Centre	1 copy
Manager, Document Exchange Centre, (for retention)	1 copy
National Technical Information Service, United States	2 copies
Defence Research Information Centre, United Kingdom	2 copies
Director Scientific Information Services, Canada	1 copy
Ministry of Defence, New Zealand	1 copy
National Library of Australia	1 copy
Defence Science and Technology Organisation Salisbury, Research Library	2 copies
Library Defence Signals Directorate Canberra	1 copy
British Library Document Supply Centre	1 copy
Parliamentary Library of South Australia	1 copy
The State Library of South Australia	1 copy
Spares	
Defence Science and Technology Organisation Salisbury, Research Library	6 copies

Department of Defence

DOCUMENT CONTROL DATA SHEET

1. Page Classification	
Unclassified	
2. Privacy Marking/Caveat (of document)	

3a. AR Number	3b. Laboratory Number	3c. Type of Report	4. Task Number	
AR-008-839	ERL-0800-RR	Research Report	DEF 93/163	
5. Document Date	6. Cost Code	7. Security Classification	8. No. of Pages	100
JUNE 1994	252	* [U] [U] [U]	9. No. of Refs.	13

Document Title Abstract

S (Secret) C (Confi) R (Rest) U (Unclass)

* For UNCLASSIFIED docs with a secondary distribution LIMITATION, use (L) in document box.

10. Title

MODULAR IMPLEMENTATION OF FEATURE EXTRACTION AND MATCHING ALGORITHMS FOR PHOTOGRAMMETRIC STEREO IMAGERY

11. Author(s)

James Kershaw and Garry Hamlyn

12. Downgrading/Delimiting Instructions

N/A

13a. Corporate Author and Address

Electronics & Surveillance Research Laboratory
PO Box 1500, Salisbury SA 5108

13b. Task Sponsor

DIO

14. Officer/Position responsible for

Security:.....

Downgrading:.....

Approval forRelease:....DERL.....

15. Secondary Release Statement of this Document

APPROVED FOR PUBLIC RELEASE

16a. Deliberate Announcement

No Limitation

16b. Casual Announcement (for citation in other documents)

[X] No Limitation [ ] Ref. by Author , Doc No. and date only.

17. DEFTEST Descriptors

Image Analysis
Image Interpretation
Matching
Algorithms
Three dimensional

18. DISCAT Subject Codes

12

19. Abstract

This paper describes the implementation of algorithms for automatically extracting and matching features in stereo pairs of images. The implementation has been designed to be as modular as possible to allow different algorithms for each stage in the matching process to be combined in the most appropriate manner for each particular problem. The modules have been implemented in the AVS environment but are designed to be portable to any platform.

This work has been undertaken as part of task DEF 93/163 "Intelligence Analysis of Imagery", and forms part of ITD's contribution to the Visual Processing research program in the Centre for Sensor System and Information Processing. A major aim of both the task and the research program is to produce software to assist intelligence analysts in extracting three dimensional shape from imagery: the algorithms and software described here will form the first part of a module for automatically extracting depth information from stereo image pairs.